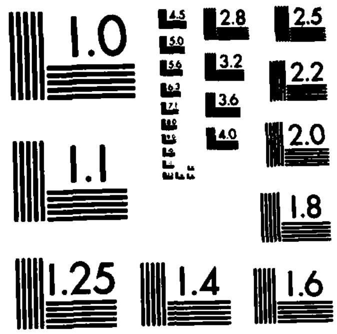
AD-A126 850 ALTERATION AND IMPLEMENTATION OF THE CP/M-86 OPERATING SYSTEM FOR A MULTI-USER ENVIRONMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA T V ALMQUIST ET AL. 1/2

UNCLASSIFIED DEC 82 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

ALTERATION AND IMPLEMENTATION OF THE CP/M-86
OPERATING SYSTEM FOR A MULTI-USER ENVIRONMENT

by

Thomas V. Almquist

and

David S. Stevens

December 1982

Thesis Advisor: U. R. Kodres

Approved for public release; distribution unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A126950	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Alteration and Implementation of the CP/M-86 Operating System for a Multi-User Environment		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Thomas V. Almquist David S. Stevens		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1982
		13. NUMBER OF PAGES 160
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CP/M-86, multi-user CP/M-86 system, table-driven CP/M-86 BIOS, AEGIS "signal processor" emulation, magnetic bubble memory, REMEX Data Warehouse.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CP/M-86 is a single user microcomputer operating system developed by Digital Research. This thesis provides a multi-user "protected" CP/M-86 based disk sharing environment consisting of four Intel iSBC 86/12A single board computers, a MBB-80 bubble memory, and the REMEX Data Warehouse 3200 memory storage unit. The REMEX houses a 14 inch Winchester hard disk and two flexible

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(continuation of abstract)

floppy disk drives providing in excess of 20 megabytes of data storage capacity. The major objective in the design of this system was to create a table-driven CP/M-86 Basic Input/Output System that could be quickly and easily reconfigured to adapt to any new hardware configuration. Once the system was operational, the REMEX hard disk could then serve as a "single processor" emulation for the AEGIS system. By making direct calls to the appropriate read/write routines, stored "radar data" could be retrieved from the hard disk for use by the other system processes.

Approved for public release; distribution unlimited

Alteration and Implementation of the CP/M-86 Operating System for a Multi-user Environment

by

Thomas V. Almquist
Lieutenant Commander, United States Navy
B.S.A.E., North Carolina State University, 1971

David S. Stevens
Captain, United States Army
B.S.E., United States Military Academy, 1974

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1982

Authors: Thomas V. Almquist

David S Stevens

Approved by: Uno R. Kodres
Thesis Advisor

Second Reader

David K. Hsiao
Chairman, Department of Computer Science

W M Woods
Dean of Information and Policy Sciences

ABSTRACT

CP/M-86 is a single user microcomputer operating system developed by Digital Research. This thesis provides a multi-user protected CP/M-86 based disk sharing environment consisting of four Intel iSBC 86/12A single board computers, a MBB-80 bubble memory, and the REMEX Data Warehouse 3200 memory storage unit. The REMEX houses a 14 inch Winchester hard disk and two flexible floppy disk drives providing in excess of 20 megabytes of data storage capacity. The major objective in the design of this system was to create a table-driven CP/M-86 Basic Input/Output System that could be quickly and easily reconfigured to adapt to any new hardware configuration. Once the system was operational, the REMEX hard disk could then serve as a "signal processor" emulation for the AEGIS system. By making direct calls to the appropriate read/write routines, stored "radar data" could be retrieved from the hard disk for use by the other system processes.

DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark.

Intel corporation, Santa Clara, California:

Intel	MULTIBUS	INTELLEC MDS
Intel 8080	Intel 8086	iSBC 86/12A
iSBC 202	iSBC 201	

Pacific Cyber/Metrixs Incorporated, Dublin, California:

Bubble-Tec	Bubbl-Machine	MBB-Bubbl-Board

Digital Research, Pacific Grove, California

CP/M	CP/M-80	CP/M-86
MP/M		

EX-CELL-O Corporation, Irvine, California

REMEX Data Warehouse

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

One of the most popular operating systems available for microcomputers today is the family of Digital Research's CP/M operating systems. They are single user systems which can be configured to interface with nearly any existing piece of hardware simply by redesigning the Basic Input/Output System (BIOS) module of CP/M. Since CP/M is a single user system, protection from other users is not normally an issue of concern with this operating system.

MP/M, also marketed by Digital Research, is a multi-user operating system which supports multiprogramming on a uniprocessor. It is basically an expanded version of CP/M. However, MP/M provides virtually no protection for user files and very little protection for memory in the event that another user's process crashes. Furthermore, when more than one user is operating under MP/M, system response time is noticeably increased.

B. PURPOSE

This thesis presents an implementation of CP/M-86 which will permit multiple users, each with his own microcomputer, to access the same peripheral devices in a manner similar to that of the MP/M operating system, but

with increased user protection. The peripherals used in this implementation are a 32K common memory board, a MBB-80 magnetic bubble memory configured as a floppy disk drive, and a Remex Data Warehouse memory storage unit consisting of a Winchester hard disk and two flexible floppy disk drives. In addition, computer performance is not compromised since each user has a dedicated INTEL 86/12A iSBC on which to operate.

The standard version of CP/M-86 requires that only the BIOS be altered to add additional hardware. While this is an excellent method to interface hardware with CP/M, it requires that the BIOS be rewritten every time the hardware configuration is changed. This process can become time consuming and is definitely prone to errors, thus discouraging frequent system reconfiguration. Therefore, in the design of this system, a major goal was to develop a BIOS which could easily be modified if it was necessary to convert from one hardware configuration to another.

This thesis was based on work accomplished in two previous theses. Michael Candelor's thesis entitled "Alteration of the CP/M Operating System" [Ref. 1] initially modified CP/M-86 to interface with the Intel i201 and i202 Floppy Disk Controllers. Michael Hicklin and Jeffery Neufeld, in their thesis "Adaptation of the Magnetic Bubble Memory in a Standard Microcomputer Environment", [Ref. 2] interfaced the MBB-80 Bubbl-Board and the i202

Floppy Disk Controller with the CP/M-86 Operating System. Although Hicklin and Neufeld claimed that their BIOS was table-driven, it was Nick Hammond who really identified that the BIOS functions could be truly table-driven [Ref. 3]. This thesis builds on the ideas contained in each of these previous works and expands upon them to create a more practical and versatile operating system which provides increased protection of the user's address space and files.

Once the system was operational, the REMEX hard disk could then be used to emulate the "signal processor" functions of the AEGIS system. Direct calls can be made to the appropriate read/write driver routines to retrieve stored "radar data" from the hard disk for use by the other emulated processes in the system.

This thesis has been organized into four major sections. The first section deals with an overview of CP/M-86 and the necessary steps required to create a new CP/M-86 system. It also describes how the BIOS interfaces with the other modules of CP/M-86 and the peripheral devices. Included in this section is a description of how the BIOS can be reconfigured into a table-driven operating system which will permit easy alterations to the BIOS if the hardware configuration should be modified.

The second section describes the hardware configuration utilized in this thesis. The memory organization of the MBB-80 Bubbl-Board is discussed and the design decisions

that were made to make the bubble memory compatible with the CP/M operating sytem are treated in some detail. The basic functions of the REMEX Data Warehouse are also described, as well as, the command packet structure and execution.

The third section is concerned with the development of a CP/M-86 operating system which will permit four single board computers to operate simultaneously while sharing the same peripherals. In this design, it is necessary to provide protection to common memory during read and write operations and to insure that each user's files are write protected with respect to all other uses.

The final section describes the tests that were conducted to evaluate system performance. In addition, the feasibility of using the REMEX hard disk to emulate the "signal processor" of the AEGIS system was explored. Measurements were made using direct calls to low-level read routines to determine the optimum skew factor for consecutive sector access operations. Also, some recommendations were made for future projects involving the REMEX Data Warehouse and the multi-user CP/M-86 operating system.

II. CP/M-86

A. THE CP/M OPERATING SYSTEM

CP/M-86 is an operating system developed for use on a single INTEL Corporation 86/12A microcomputer. CP/M is supplied with a number of built-in utility commands as well as transient utilities such as the assembler (ASM86.CMD) and the Dynamic Machine Language Program Debugger (DDT86.CMD). These are described in detail in Digital Research publications. [Refs. 4 - 6]

The CP/M operating system itself is modularized to permit easy adaption of CP/M to any hardware configuration. The three modules are the Console Command Processor (CCP), the Basic Disk Operating System (BDOS) and the user configurable Basic Input/Output System (BIOS). The first two modules are supplied by Digital Research as a single hex file entitled CPM.H86. This file contains all the code necessary for processing commands entered at the console and for handling all logical file and disk management functions. The source code for a skeleton BIOS is also provided which the user can alter to suit his individual hardware requirements. Once the BIOS has been modified, it is assembled and then concatenated with CPM.H86. The resulting hex file, CPMSYS.H86, is converted to an executable file by the use of the CP/M utility program

```
1.  USER  BIOS.A86    ==>  ASM86.CMD  ==>  USER BIOS.H86

2.  CPM.H86 + USER BIOS.H86    ==>  PIP.CMD  ==> CPMSYS.H86

3.  CPMSYS.H86  ==>     GENCMD.CMD      ==>  CPMSYS.CMD
                      (8080 CODE[A40])

4.  CPMSYS.CMD  ==>      PIP.CMD     ==>  CPM.SYS
                    (rename on new disk)
```

Figure 2.1
Steps for Creating CPM.SYS

GENCMD.CMD. Finally, this file is renamed CPM.SYS and placed on a diskette for use. This process is shown in Figure 2.1. Details concerning the operation of GENCMD.CMD, LDCOPY.CMD and PIP.CMD can be found in the "CP/M-86 Operating System Guide". [Ref. 6]

CP/M-86 supports programs written in three memory models: the 8080 Model, the Small Model, and the Compact Model. All three memory models are described in detail in Reference 5. The model used in this thesis is the 8080 Model because it supports programs which have code and data areas intermixed and which normally have single segments of 64K bytes or less.

B. LOADING CP/M-86

The file CPM.SYS is too large to fit onto the first two tracks of a normally-formatted diskette. Thus, a boot loader must be placed on these tracks and loaded into memory by the cold start loader. This boot loader program will

then bring the main CP/M operating system into memory and pass control to it.

The loader program is distributed by Digital Research in three separate modules and is basically a subset of the entire CP/M system. The modules are the Loader Console Command Processor (LDCCP.H86), the Loader Disk Operating System (LDBDOS.H86), and a user configurable Loader Basic Input/Output System (LDBIOS.A86) which is almost identical to the system BIOS. The primary differences deal with the physical memory location of the loader, the interrupt structure and the BIOS offset address within the CP/M system. Assembly of the loader BIOS is controlled by a conditional assembly switch provided in the skeleton BIOS, which is listed in Appendix E of Reference 6. The steps needed to obtain a loader BIOS are essentially the same as for creating the CPM.SYS. The exact steps are shown in Figure 2.2.

```
1. USER LDBIOS.A86  ==>  ASM86.CMD  ==>  USER LDBIOS.H86

2. LDCCP.H86 + LDBDOS.H86 + USER LDBIOS.H86 ==>  PIP.CMD
                     ==>  LOADER.H86

3. LOADER.H86  ==>  GENCMD.CMD  ==>  LOADER.CMD
                   (8080 CODE[A400])

4. LOADER.CMD  ==>  LDCOPY.CMD  ==>  LOADER.CMD
              (load on tracks 0 and 1)
```

Figure 2.2
Steps For Creating Boot LOADER.CMD

C. BOOTSTRAPPING THE iSBC 86/12A

From the monitor of the iSBC 86/12A, the CP/M system loader program located on tracks 0 and 1 of the disk, can be accessed via the bootstrap or cold start loader program. This program is located in ROM or EPROM on the iSBC 86/12A board itself. Thus, for each separate device from which the system is to be booted, a new cold start loader program must be written and then burned into ROM. Finally, this ROM must be mounted on the iSBC 86/12A board where it can be accessed by the monitor program.

Currently, two cold start loader programs are available for the iSBC 86/12A. One allows the system to be booted from either the single or double density Intel MDS floppy disk drive system by executing the command GFFD4:0 from the iSBC 86/12A 957 monitor program. When this command is executed, the program in the ROM will go out to tracks 0 and 1 of the floppy diskette and attempt to bring into memory the CP/M system loader program. Once loaded into memory, the cold start loader will then transfer control to the loader which in turn will locate the CP/M system (CPM.SYS) on the disk and load it into memory. Finally, the system loader will relinquish control to the CP/M operating system. The source code for this bootstrap program is listed in Appendix C of Reference 1.

The second program allows bootloading from the MBB-80 bubble memory device by issuing the command GFFD4:4.

Currently this last command can only be used when operating on the iSBC 86/12A which is labeled #1, as it is the only computer with an EPROM that contains the cold start loader for the bubble memory. The source code for this program, which was developed by Hicklin and Neufeld, can be found in Appendix D of Reference 2.

This thesis uses the bubble memory to initially boot the system. Therefore, a new cold start loader program or CP/M system loader program did not have to be developed. All that is required to change the operating system that will be loaded is to place a new CP/M system (CPM.SYS) on the bubble memory storage device.

The loader program placed on tracks 0 and 1 of the bubble memory used for loading the CP/M operating system is entitled MB80LDR.CMD. This file is created by following the steps indicated in Figure 2.2 utilizing MB80BICS.A86 as the source file with the loader conditional assembly switch set to true.

D. DISK PARAMETER TABLE

The CP/M-86 operating system as marketed by Digital Research is considered a table driven system since all characteristics for each I/O device is placed in a table called the Disk Parameter Table which can handle up to sixteen separate devices. This table defines the logical organization of the physical storage media for the BDOS file management functions and must be included in every BIOS.

A disk definition statement is required for each physical device and consists of a sequence of words which define the characteristics of a device. Figure 2.3 shows the format of a disk definition statement. These statements are then used to generate the Disk Parameter Table by executing the utility program entitled GENDEF.CMD [Ref 6, p.72]. The file created by this program must be included in

```
DISK DEF: dn, fsc, lsc, [skf], bls, dir, cks, ofs, [0]

        where
         dn is the logical disk number (0 to 15)
         fsc is the first physical sector number (0 or 1)
         lsc is the last logical 128 byte sector number
         skf is the optional skew factor
         bls is the data allocation block size
         dsk is the disk size in bls units
         dir is the number of directory entries
         cks is the number of "checked" directory entries
         ofs is the track offset to logical track 0
           (normally 2 as track 0 and 1 contain the loader)
         [0] is the optional 1.4 version compatibility flag
```

Figure 2.3
Format of Disk Definition statement

the BIOS using an "include" statement. The file which contains the disk definition statements for this thesis is labeled CPMMAST.DEF and used to generate a Disk Parameter Table which is located in the file called CPMMAST.LIB. These two files can be found in Appendices G and H.

To create a disk definition statement for the table, the characteristics for the device must be known. This information is usually located in the technical manuals for

the given device. For example, the disk definition statement used for the REMEX Winchester hard disk was:

DISKDEF 3,1,156,Ø,16384,255,128,Ø,1.

The first "3" indicates that the hard disk is CP/M's logical drive number "3" and can be accessed via the "D:" command from within CP/M.

The next two numbers correspond to the first and last logical sector numbers for the Winchester hard disk as seen by CP/M. The actual physical sectors for the hard disk are numbered from 1 to 39, each containing 512 bytes. Since CP/M requires the number of logical 128 byte sectors, 39 is multiplied by 4 to produce 156 logical sectors of 128 bytes. The actual mapping from the logical to the physical sectors is accomplished in the blocking and deblocking subroutines located in the code for the REMEX hard disk (RXHARD.A86) and is described in more detail in the Chapter IV.

The REMEX technical manual does not indicate what the most effective skew factor is, thus zero was chosen because it was required by the blocking and deblocking routines. However, an optimal skew factor may be determined experimentally when the REMEX hard disk is used to emulate the "signal processor" of the AEGIS system. If so, the blocking/deblocking routine will have to be modified at that time.

The "bls" parameter specifies the number of bytes allocated to each data block. This number can be 1024, 2048, 4096, 8192, or 16,384. When larger block sizes are used, each directory entry can address more data. This reduces the amount of work that the BIOS must do, resulting in reduced system response time. Therefore, a block size of 16,384 was chosen.

The "dsk" specifies the total disk size in terms of data blocks. It is derived by dividing the total byte capacity of the disk by the data block size. In this implementation, the Winchester disk contains approximately 20 megabytes of data storage which is subdivided between four separate heads. Thus 4,193,280 bytes are allocated to the "D:" drive and this figure is divided by 16,384 to produce 255 data blocks.

The next figure, 128, indicates the number of directory entries that are permitted on this drive.

The "cks" term determines the number of directory items to be checked on each directory scan and is primarily used for detecting changed disks during system operations. As the Winchester disk is permanently mounted, a value of zero was chosen for this parameter.

The "ofs" value determines the number of tracks to be skipped when accessing the disk. In essence, it reserves tracks for permanent storage. Track 0 is reserved since the Remex requires it for internal system use and errors will

occur if an attempt is made to access it. On a floppy disk, this value is usually two as tracks 0 and 1 are normally reserved for the loader program.

E. THE STANDARD BIOS

The BIOS for CP/M-86 always begins at an offset of 2500 hex from the beginning of the CP/M-86 operating system. At this location are twenty-one entry points used by the CCP and the BDOS to gain access to the BIOS functions. These entry points form a jump vector to other subroutines in the BIOS which contain the necessary code to interface with each hardware device.

There are three types of functions in the BIOS: system initialization/reinitalization, simple character I/O and disk I/O. Several of these functions are normally not implemented in most microcomputer systems, while others require extensive and quite different code implementations for each separate device. The BIOS also contains the Disk Parameter Tables which represent the physical description of the disk drives. Finally, located at the end of the BIOS, there is a scratchpad area for certain BDOS operations. Figure 2.4 shows the memory map of the BIOS.

In order to simply access a diskette, several functions located within the BICS may have to be performed. For example, to access the directory of a diskette, the BDOS will require the following functions to be performed by the BIOS: SELDSK, HOME, SETTRK, SETSEC, SETDMA, SETDMAB and

CS, DS, ES, SS:	CONSOLE COMMAND PROCESSOR & BASIC DISK OPERATING SYSTEM
CS + 2500H:	BIOS JUMP VECTOR
CS + 253FH:	BIOS SUBROUTINES
	DISK PARAMETER TABLES
	UNINITIALIZED SCRATCH RAM

Figure 2.4
Memory Map of the Standard BIOS

READ. [Ref. 6: p.60] For each function executed, the BIOS will have to determine which physical device is being accessed and then jump to or call the subroutine which contains the code for that specific device. For example, suppose a simple READ function is required by the BDOS. It will initiate a call to the BIOS READ entry point which in turn will vector the call to the READ subroutine. Here the BIOS will determine which physical device corresponds to the CP/M's logical drive and then jump to the appropriate code to read data from that specific device. (See Figure 2.5)

This procedure is very logical and makes it easy for a user to implement his specific device dependent

```
    CCP   ---
             |----> call to BIOS to Read Device  #2 -->---
    BDOS  ---                                             |
                                                          |
  -----------------<--------------<------------------------
  |
  |  BIOS: jmp init
  |        jmp write
  ------>  JMP READ ---->------
            .                 |
            .                 |
           jmp wboot          |
                              |
 -------<-----------<----------
 |
 |         init:
 |            code for initializing all devices
 |         ret
 |
 |         write:
 |            code for writing to all devices
 |         ret
 |
 ------>---READ:
              determine device
              jmp read_device #1
              JMP READ_DEVICE #2 --->-----
              jmp read_device #3         |
                                         |
 -------<-------------<-------------------
 |
 |                 read_device #1:
 |                      code for reading device #1
 |                 ret
 |
 ------->--------- READ_DEVICE #2:
                        CODE FOR READING DEVICE #2
                   RET

                   read_device #3
                        code for reading device #3
                   ret
                       .
                       .
                       .
```

Figure 2.5
Path of CCP or BDOS Function
Call in the Standard BIOS

code. However, problems arise if the hardware configuration must be altered. Everytime the configuration changes, the code for each function in the BIOS must be rewritten. This can be a time consuming task. In addition, assumptions made concerning the implementation of one configuration may lead to errors in another configuration should those assumptions no longer be valid. These errors may also be extremely difficult to locate and correct since all code is usually intermixed and the exact order that the CCP and BDOS call various functions in the BIOS is not known to the user.

F. BIOS ALTERATION

Hicklin and Neufeld attempted to develop a table driven BIOS. In a manner of speaking they succeeded. However, the only devices that are permitted in their device table are additional Intel MDS double density disk drive systems and MBB-80 bubble memory storage devices. Attempting to integrate another device such as the REMEX Data Warehouse, leads to the same problems which were mentioned earlier.

To alleviate these problems, a completely table-driven BIOS was developed in which only minor and straight-forward changes would have to be made in order to change hardware configurations. This was accomplished by extracting out all the device-dependent functions of the BIOS into separate files for each unique device. Specifically, these functions were INIT, SELDSK, HOME, SELTRK, SELSEC, READ, and WRITE. Functions such as WBOOT are not dependent upon a particular

device and do not have to be extracted, while functions such as PUNCH and READER are not implemented.

In the hardware configuration for this thesis, three separate files were required. These were MB80DSK.A86, RXFLOP.A86, RXHARD.A86. These files each contain the necessary code to execute the seven device-specific functions for the MBB-80 bubble storage device, the Remex

```
                    |-------------------------|
CS, DS, ES, SS:     |                         |
                    |    CONSOLE  COMMAND     |
                    |       PROCESSOR         |
                    |           &             |
                    |  BASIC DISK OPERATING   |
                    |        SYSTEM           |
                    |                         |
                    |-------------------------|
     CS + 2500H:    |   BIOS JUMP VECTOR      |
                    |-------------------------|
     CS + 253FH:    |                         |
                    |   BIOS SUBROUTINES      |
                    |                         |
                    |-------------------------|
                    |                         |
                    |  INCLUDE LABEL TABLES   |
                    |  INCLUDE DEVICE #1      |
                    |  INCLUDE DEVICE #2      |
                    |  INCLUDE DEVICE #3      |
                    |          .              |
                    |          .              |
                    |  INCLUDE DEVICE #16     |
                    |                         |
                    |-------------------------|
                    |    DISK PARAMETER       |
                    |        TABLES           |
                    |-------------------------|
                    |    UNINITIALIZED        |
                    |     SCRATCH RAM         |
                    |-------------------------|
```

Figure 2.6
Memory Map of the Table-Driven BIOS

floppy disk drives, and the Remex hard disk, respectively. An additional file, CPMMAST.CFG, is also now required. It contains tables of labels which correspond to the physical memory location of the seven functions for each device used in a given hardware configuration. The label tables used in this thesis can be found in Appendix C. Figure 2.6 shows the memory map of the table driven BIOS. In the BIOS, the assembly language instruction "include" is used to incorporate the label tables and device-specific code for the seven functions into the system.

For example, when a call is made to read Device #2 from the CCP or the BDOS, the call is vectored as was done before through the jump vector to the READ subroutine of the BIOS. However, after determining the physical device to be accessed, instead of jumping directly to the desired code, a call is now made to the device specific code located in the included device's A86 file via the Read Table which is located in the file CPMMAST.CFG. The final address of the call is determined by the offset of device number into the Read-Table, which provides the label or 16-bit address of the actual code needed for reading Device #2. (See Figure 2.7)

To alter the hardware configuration, only one line in the BIOS must now be changed for each device, that being the corresponding "include" statement. The other changes which are required, are located in the label tables and the Disk

```
   CCP & BDOS  ---> call to BIOS to Read Device #2 -->---
                                                         |
 ----------------------<---------------<-----------------
 |
 |  BIOS: jmp init
 |        jmp write
 ------>  JMP READ ---->------
           .                 |
          jmp wboot          |
                             |
-------<-------------<--------
|         init:
|              call to init label table
|              ret
|         write:
|              call to write label table
|              ret
------>---READ:
               determine device
               CALL READ_TABLE [offset device #2] --->----
               ret                                       |
                                                         |
-------<----------------<------------------<--------------
|        INCLUDE LABEL TABLE
|              init_table:
|                  labels
|              write_table:
|                  labels
|              READ_TABLE:
|                  read_device #1
-------->  ----------READ_DEVICE #2--->----
                   read_device #3         |
                                          |
-----------<-------------------<-----------
|        include device #1
|              code for seven device specific functions
|
|        INCLUDE DEVICE #2
|              init_device #2
|                  code for initializing Device #2
|              write_device #2
|                  code for writing to Device #2
-------->-------READ_DEVICE #2
                    CODE FOR READING DEVICE #2
                         .
                         .
```

Figure 2.7
Path of CCP or BDOS Function
Call in Table-Driven BICS

Parameter Tables. For each device included in the BIOS, there must be a corresponding label for an abstracted function. These labels must be correctly ordered and properly identified. Naturally, when hardware is implemented into the system for the first time, the initial code for performing the seven device-specific functions must be written. But once written, the new device can be added or deleted from the operating system with very little effort. The fact that all code for each device is completely independent of other devices, aids in detecting, locating and correcting errors. Actual experience has shown that once the code for a device has been written, going from one hardware configuration to another can be accomplished in under twenty minutes.

III. HARDWARE

A. GENERAL HARDWARE CONFIGURATION

The hardware configuration utilized in this thesis consists of four iSBC 86/12A Single Board Computers, a MBB-80 Bubbl-Board, a 32K byte common memory board, and the REMEX Data Warehouse memory storage device with Multibus Interface Card Assembly. The components are all Multibus compatible and were placed in an iCS-80 Industrial Chassis for system operation. Figure 3.1 depicts the physical hardware configuration. Table 3.1 describes the logical-to-physical mapping between the CP/M representation of the system and the actual physical hardware.

```
-------------     ---------     ------------
| 32K COMMON |    | BUBBLE |    |   REMEX    |
|   MEMORY   |    | MEMORY |    | INTERFACE  |======
-------------     ---------     ------------     ||
     ||              ||              ||          ||
     ||              ||              ||       --------
------------------------------------------    | REMEX |
|               MULTI- BUS               |    | DATA  |
------------------------------------------    | WARE- |
    ||         ||          ||         ||      | HOUSE |
    ||         ||          ||         ||      --------
 --------   --------   --------   --------
|  iSCB  | |  iSCB  | |  iSCB  | |  iSCB  |
| 86/12A | | 86/12A | | 86/12A | | 86/12A |
 --------   --------   --------   --------
```

Figure 3.1
Physical Hardware Configuration

Table 3.1
Logical Hardware Configuration

CP/M's Logical Device Number	Actual Drive	Actual Physical Device
Ø	A:	MBB-8Ø Bubble Memory
1	B:	Remex Floppy Disk Drive
2	C:	Remex Floppy Disk Drive
3	D:	Remex Hard Disk Head Ø
4	E:	Remex Hard Disk Head 1
5	F:	Remex Hard Disk Head 2
6	G:	Remex Hard Disk Head 3

B. INTEL 86/12A SINGLE BOARD COMPUTER

The Intel iSBC 86/12A Single Board Computer is a complete computer system constructed entirely on a single Multibus-compatible circuit board. It is designed to operate as a standalone system, a bus master in a single bus master system, or a bus master in a multiple bus master system. The board itself contains an Intel 8Ø86 16-bit microprocessor, 64K bytes of dynamic RAM memory, 16K bytes of EPROM memory, both serial and parallel I/O ports, a programmable timer and interrupt controller, and a Multibus interface controller.

Onboard RAM memory is located between Ø and Øffffh and the EPROM between FFCØØh and FFFFFh within the 1-Megabyte address space available to the Intel 8Ø86 microprocessor.

If the local processor attempts to address memory outside of these ranges, a Multibus access will result. The onboard RAM is dual-ported, and therefore is accessible to the local processor via an internal bus, as well as, to any external Multibus master via the Multibus. In this latter case, the onboard RAM is operating in the RAM-Slave mode. Any collisions that result when the RAM is simultaneously accessed by the local CPU and the Multibus are resolved by hardware in favor of the local CPU.

While the location of RAM relative to the local processor is fixed between 0 and FFFFh, it can be switch-and-jumper configured into any 128K segment of the 1-Megabyte address space relative to the Multibus. In addition, none or all of the onboard RAM, in segments of 16K, may be reserved strictly for local CPU use. Since the major objective of this implementation was to produce a CP/M-based multicomputer system in which each computer operates totally independently of the others, each iSBC 86/12A was configured to make all of the onboard RAM inaccesible to the Multibus.

C. MBB-80 BUBBLE MEMORY STORAGE DEVICE

1. General Description

The MBB-80 Bubbl-Board is a complete bubble memory storage device designed to be compatible with all 8- and 16-bit microcomputers that utilize Intel's Multibus architecture. The board consists of eight (8) TIB0203

bubble devices and the necessary control, buffering, and Multibus interface logic. The host CPU interfaces with the MBB-80 controller via memory-mapped I/O utilizing any sixteen (16) consecutive user-defined addresses within the 1-Megabyte system address space. These sixteen (16) addresses correspond to the sixteen (16) registers in the bubble memory controller that are utilized in support of the following controller primitive commands:

Fill Buffer	Read Multiple Pages
Empty Buffer	Initialize
Write Single Page	Read Status
Read Single Page	Enable/Disable Interupts
Write Multiple Pages	Reset

2. Read/Write Logic

Read and write operations with the MBB-80 are accomplished by specifying a particular bubble device number and page number (18 bytes) to read from or write to. The MBB-80 controller provides the ability to read or write in either a single- or multiple-page mode by using a byte-by-byte transfer into a FIFO buffer located on the MBB-80 board itself. The single-page mode can be implemented in a straight-foward manner without the need for addtional supporting hardware or software. However, the multiple-page mode requires that certain timing requirements must be adhered to by the host CPU when communicating with the MBB-80 controller. During a data transfer, the host must

respond to interrupts generated by the MBB-80 every 160 microseconds which signal the completed transfer of one byte of information in a multi-byte transfer. These interrupts can be generated on the Multibus and handled by the Programmable Interupt Controller (PIC), or the host CPU can poll the controller interrupt register (offset 0fh) to determine if an interrupt has occurred. The single- and multi-page polled modes were implemented by Hicklin and Neufeld [Ref. 2]. The final version of their system utilized the multi-page polled mode and this was subsequently employed in this implementation.

3. CP/M-86 Compatibility

In order to effect a data transfer, the MBB-80 controller must be given a device and initial page number to locate the position where the data will be read from or written to. On the other hand, CP/M uses a track and sector number to access data during a disk access. Therefore, a mapping must be made from the CP/M track and sector number to MBB-80 device and page numbers if the CP/M operating system is going to be used to access data on the MBB-80 Bubbl-Board. Hicklin and Neufeld [Ref. 2] decided to use the bubble page number as the smallest addressable unit for each data transfer and the basis for the MBB-80 memory organization. Since each physical bubble page is eighteen (18) bytes long, a logical CP/M sector of 128 bytes consists of eight (8) bubble pages of which the last sixteen (16)

bytes on the last page are not used (i.e. wasted). Therefore, the 640 bubble pages per device are mapped into 80 logical CP/M sectors per device. Futhermore, it was decided that each MBB-80 "track" would consist of 26 sectors which corresponds to the number of sectors per track on a normally-formatted single-density floppy disk. Another design decision was that all MBB-80 tracks would be completely contained on a single bubble device. Since there are 26 CP/M sectors per track and 80 sectors per bubble device, this results in three (3) tracks per bubble device with two (2) sectors not used or wasted on each device. Therefore, based on these design decisions, the total capacity of the MBB-80 Bubbl-Board is 78K bytes on 24 tracks (8 devices x 3 tracks per device) with a total of 14k bytes wasted. Hicklin and Neufeld's final memory organization for the MBB-80 is shown in Figure 3.2. Dispite its inefficiency, this configuration was adopted for this implementation since the principal function of the bubble memory is to provide a convenient method of booting CP/M-86 on our master iSBC 86/12A. Hammond [Ref. 3] has shown that there is a more efficient way to organize the MBB-80 in his work on utilizing the MBB-80 as a shared resource in a multi-microcomputer system. However, this would have necessitated the design and implementation of a new bootstrap loader program to be placed in the iSBC 86/12A

EPROM and was not judged to be of significant importance for this implementation.

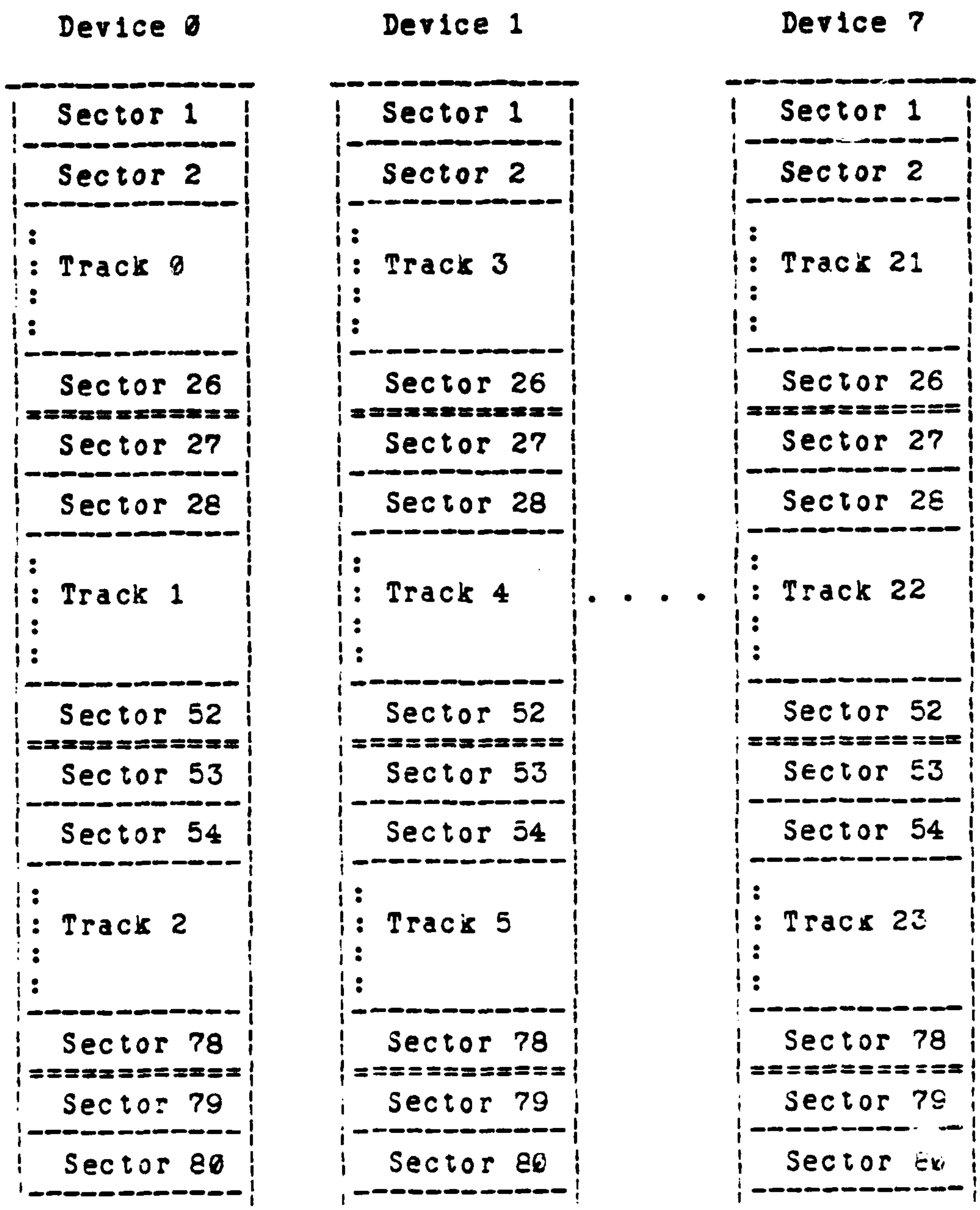


Figure 3.2
MBB-8Ø Logical Storage Organization

D. REMEX DATA WAREHOUSE

1. General Description

The REMEX Data Warehouse is a mass storage memory unit containing a fixed Winchester disk drive, two (2) flexible diskette drives (single- or double-sided), and a microprocessor controller that services all drives. The memory capacity of the fixed disk is approximately 20 megabytes and the flexible diskettes can be formatted to contain up to two (2) megabytes of storage. IBM standard FM encoding is used for the single density floppy diskette while MFM encoding is utilized for the double density diskette and the hard disk.

The fixed disk is a 14 inch enclosed disk utilizing Winchester technology and is composed of two recording surfaces. Each surface has two (2) recording heads which can each access a total of 213 tracks. Each track can contain up to 24K bytes of information. However, only 210 tracks can be referenced for normal read/write operations. The hard disk sector size is switch-selectable to either 128, 256, 512, or 1024 bytes per sector. The total storage capacity for the various sector sizes is shown in Table 3.2. In addition, the floppy diskette controllers are also switch-selectable to handle either single or double density diskettes. <u>It is extremely important that these switch settings correspond exactly to the actual format of the hard</u>

Table 3.2
REMEX Hard Disk Sector Selections

Sector Size	Sectors/Track	Capacity
128	104	10.7M bytes
256	67	14.4M bytes
512	39	16.8M bytes
1024	21	18.1M bytes

disk and diskette for the read/write operations to function correctly.

The REMEX Data Warehouse (RDW) is designed to transfer all data and command structures to and from the host computer via direct memory access (DMA). To initiate a RDW operation, the host computer builds a command packet within its local memory. This packet contains all the information necessary to effect an RDW operation. The host then sends the address of the command packet to the RDW via an interface board utilizing programmed I/O. When the RDW is ready to accept packets, it inputs the command packet via DMA, performs the required function, and transfers any data via DMA. When the function is complete, the RDW indicates this by noting it in the command packet status word or by generating an interrupt on the Multibus. Packets can be queued in the RDW up to a maximum of eight.

Some other important features of the RDW include:

-- Dynamic data buffering (2K x 16 bit buffer)

allows a continuous transfer under varying CPU conditions.

-- Dynamic buffer protects against data overrun and underrun preventing loss of data without host computer intervention.

-- Allows data transfers in large blocks of up to 64K words with a single command. Heads are automatically advanced as necessary.

-- Automatically seeks to track(s) required in command packet.

-- Permits chaining packets together in noncontinuous memory.

-- Ability to format entire disk with a single command.

--Automatic verification and assignment of alternate tracks to cover bad tracks.

2. Command Packet Organization

The basic structure of the command packet is shown in Figure 3.3. Word 0 is composed of a modifiers section, a function code block , and a logical unit section. The function code block specifies which of the six (6) particular REMEX functions is to be performed. These functions are Read, Write, Write ID and Record, Copy, Format, and Maintenance.

```
                    Bit Number

       15               8 7             4 3          0
       ------------------------------------------------
word 0 |     Modifiers    |   Function   | Logical Unit |
       ------------------------------------------------
     1 |                 Status Word                    |
       ------------------------------------------------
     2 |                 -----------                    |
       ------------------------------------------------
     3 |                 -----------                    |
       ------------------------------------------------
     4 |                 -----------                    |
       ------------------------------------------------
     : |                                                |
       ------------------------------------------------
     : |                                                |
       ------------------------------------------------
     N |                  Last Word                     |
       ------------------------------------------------
```

Figure 3.3
Command Packet Description

A program in the RDW interprets the function number and determines how many words are required for each specific command packet. The modifiers section contains information on packet chaining, program control interrupts, disabling of error routines and an "end" marker which specifies a single packet or the last packet in a packet string. The logical unit can be either 0, 1, or 2. A zero always corresponds to the hard disk. However, the floppy diskette drive can be operator-configured to respond to either logical unit number 1 or 2. This is accomplished by the Device Logical Unit Switch located on the front panel of the RDW.

The status word is divided into the least significant bits (0-7) and the most significant bits (8-15). Each of the least significant bits, when set to (1),

Table 3.3
REMEX Error Codes

Bit No.	Description
0	Normal Completion
1	Not Assigned
2	Controller Error
3	Drive Error
4	CRC Error
5	Illegal Packet
6	Bad Track During Format
7	Not Assigned

represents a particular status which is indicated in Table 3.3 Bits 8- 15 represent the hex code that corresponds to the error definitions given in Table 3-6 of Reference 7.

Words 2 through N are function dependent and the number of words per command packet varies widely between RDW operations. In the version of CP/M developed in this thesis, only the Read/Write function are implemented and are used to access and transfer data. However, additional utility programs were written which utilize the other functions to format the hard disk (RXFORMAT.CMD) and to execute the built-in maintenance programs (RXMAINT.CMD) of the RDW.

The format of the Read/Write packet is shown in Figure 3.4. The description of these two operations is identical except that in a read operation a one (1) is

Bit Number

Word	15 — 8	7 — 4	3 — 0
Word 0	Modifiers	Function	Unit
1	Status Word		
2	Track Number		
3	Head Number	Sector Number	
4	Memory Address of Data (16-bit)		
5		Ext Memory Addr Bits	
6	Transfer Word Count (# of 16-bit words)		

Figure 3.4
REMEX Read/Write Packet

placed in the function code block of packet word 0 and for a write operation a two (2) is used. Both operations are permitted in blocks of up to 64K words. Any head switching or advancing which may be required is automatically performed by the RDW disk controller.

RDW track numbers are assigned from 1 to 210 for normal data transfer operations. Track 0 is always reserved for a loader or system program and can not be addressed during a normal read or write operation without generating an error. Presently, the hard disk is formatted for 512 bytes per sector which corresponds to 39 sectors per track. Head numbers for the four RDW heads run from 0 through 3. Data addresses are a 24-bit representation of the 20-bit address structure supported by the iSBC 86/12A and Multibus architecture. The transfer word count is the number of 16-

bit words that are to be transferred. For accessing the hard disk, a transfer word count of 100h was placed in the packet built by CP/M. This figure corresponds to a single sector (512 bytes) or 256 16-bit words on the hard disk, which is equivalent to the CP/M-86 Operating System view of 512 8-bit words.

3. Multibus Interface Card Assembly

The command packets are sent to the RDW via a Multibus Interface Card Assembly. The interface contains all the necessary buffers, registers and control logic required for the transfer of data, status, addresses and commands between the REMEX Data Warehouse and the iSBC 86/12A Single Board Computer. The interface operates in both a programmed I/O mode and a DMA mode. All data, status, and commands are transferred by DMA, while packet addresses and the interface Command/Status information are transferred via programmed I/O. During these transfers, the Multibus Interface acts as a bus master in the DMA mode and as a bus slave in the programmed I/O mode [Ref. 8]. Registers are provided for data, packet address holding, and DMA addresses. A DMA address counter (20 bits) allows memory addressing of up to 1-Megabyte. Control logic for DMA, bus timing, interrupt control and device address selection is also provided. Selection switches are available to alter the interface base address, interrupt priority level, and the

DMA throttle which governs how long the interface must wait between DMA transfers.

In the programmed I/O mode of operation, the Multibus Inteface responds only to I/O port addresses. Switches, as mentioned above, are used to set the base interface port address. The standard addresses for the Command/Status Register are port address Ø7Ø (least significant byte) and port address Ø71 (most significant byte). The standard addresses for the Packet/DMA Register are port addreses Ø72 and Ø73. A more thorough description of the contents of these registers is given in Table 3-2 of Reference 7.

The DMA Throttle Select is used to select the number of Multibus accesses that must be completed between consecutive DMA transfers by the Multibus Interface. A selectable range of Ø-15 transfers is provided. The standard is 1 host Multibus cycle between interface DMA cycles. This is contrary to the explanation given in Section 2.3.3 of Reference 7. In this section, the DMA throttle is presented in terms of "number of processor cyles" instead of Multibus accesses.

4. Command Packet Execution

To execute an operation contained in a command packet, the host computer must first test the Packet Address Ready Flag (port Ø7Ø) which indicates whether the RDW is ready to accept and process command packets. If this flag

is set (1), the host loads the extended address bits (bits 17-20) of the command packet into the Command/Status Register (port 070). Then the least significant byte followed by the most significant byte of the 16-bit address of the command packet must be loaded into the Packet/DMA Register (ports 072 and 073 respectively). This sequence must be followed exactly because once the most significant byte is loaded into port 073, the interface board signals the RDW that the address is complete and ready to be transferred.

Upon receiving this signal, the RDW will read the address which was placed in the ports of the interface board, fetch the command packet located at that address, and perform the operation specified in the function code block of the packet. When the operation is complete, an entry is made into the command packet status word (word 0) indicating the success or failure of the operation.

E. ICS-80 INDUSTRIAL CHASSIS

The iCS-80 Industrial Chassis consists of four (4) four-slot iSBC 604/614 Cardcages, four fans, a power supply, a control panel and a 19" RETMA (Radio-Electronics-Television Manufacturers Association) -compatible chassis. The control panel consists of an on/off/lock key switch, interrupt and reset pushbuttons, and halt/pwr on/run LED's.

The development system was designed to support a modular microcomputer-based system. Any combination of plug-in

modules which are Multibus-compatible may be installed including single board computers, memory expansion boards and peripheral interface boards. The iSBC 604 Cardcage can accomodate four (4) iSBC circuit boards and has an external plug which allows additional iSBC 614 Cardcages to be added to the chassis. The laboratory system used in support of this thesis is composed of a single iSBC 604 Cardcage and three (3) iSBC 614 Cardcages which allow a total of 16 circuit board slots. These cardcages comprise a backplane assembly that conforms to the Intel Multibus specifications and provides slots for both Mutibus master and slave boards. The master slots are odd-numbered and the slave positions are even-numbered for easy reference.

A master board is one which is capable of acquiring and controlling the Multibus, while a slave board can only be referenced by commands on the Multibus (i.e. memory expansion boards). The iCS-80 Chassis can be used with master boards operating in either a serial or parallel priority resolution scheme. In the serial mode, Multibus access contention is resolved by the board placement within the cardcage. However, an external priority resolver network is required to implement the parallel priority scheme. In this implementation, a random priority network is used to arbitrate the contentions for the Multibus. Most importantly, one of the above priority resolution schemes must be implemented or the interaction among the iSBC boards

in the cardcages will not be correct. For further information consult References 9, 10, and 11.

IV. SYSTEM DEVELOPMENT

A. INITIAL EFFORTS

1. Program Development System

During the initial stages of this thesis, it was planned to expand the work done by Hicklin and Neufeld [Ref. 2] to incorporate the REMEX Data Warehouse memory storage unit. They had developed a reconfigurable "table-driven" CP/M-86 BIOS that supported the MBB-80 Bubbl-Board and the Intel i202 double-density floppy disk controller. It was initially believed that other I/O peripheral devices could be easily included in this BIOS with a minimum of effort. Within the proposed development system, the MBB-80 would serve as the principal storage medium for newly designed programs and would provide an easy method of booting Hicklin and Neufeld's CPM.SYS within the iCS-80 chassis.

However, this development strategy had several deficiencies. Utilizing this hardware/operating system configuration, program development would be limited to the MDS or iCS-80 systems and the CPM-86 utility programs which they supported. Presently, the only compatible text editor available is the text editor distributed by Digital Research, ED.CMD. This editor is very primitive, extremely hard to use, and completely unsatisfactory for extensive

program development. Therefore, an alternative development system was required.

It was decided to use the WORDSTAR text editor on the MP/M Multi-user System to create the needed software programs. This system provided several advantages over the MDS system. First, WORDSTAR offers functions which would significantly increase productivity and allow errors to be quickly corrected. Second, MP/M-compatible versions of ASM-86 and GENCMD utilities would enable programs to be written, assembled, corrected, and converted into executable CMD files prior to their transfer to the bubble memory. Third, since the MP/M system is a multi-user system, it did not present the availability problems associated with the single-user systems such as MDS.

Ultimately, this software development scheme also proved to be unsatisfactory, as numerous steps had to be taken to move an assembled program from the MP/M system to the MBB-80 board. Since only MP/M and MDS single density diskettes were compatible, assembled programs first had to be transferred from the MDS single density system to the MDS double density system using the laboratory utility program SDXFER.COM. This required that the MDS double density system be configured with an Intel 8080 processor. Once the program was transferred to a double density diskette, the MDS double density system had to be reconfigured for use with the MBB-80 bubble board and an iSBC 86/12A. After

reconfiguring, the program could now be transferred from the double density diskette to the bubble memory. At this point the MBB-80 was physically moved to the iCS-80 chassis. Finally, the operating system could be loaded and the program executed under DDT86.

Besides being time consuming, the above process monopolized much of the laboratory's equipment. Thus, if the equipment needed to make the transfer was in use, program testing could not be carried out. However, initially, it was the only method available and therefore had to be employed.

2. Verify MBB-80 Operation

The objective of this section was to verify the proper operation of the CPM.SYS developed by Eicklin and Neufeld. The double density MDS system was configured with a single iSBC 86/12A (#1), the MBB-80 bubble memory, and the 1202 Floppy Disk Controller. The system was successfully booted from the 957 Monitor in accordance with the procedures given in Reference 11 by executing the command GFFD4:0. However, the bubble memory could not be accessed using any of the CP/M built-in commands. After inspection of the BIOS, it was evident that the final version of the CP/M-86 BIOS submitted did not support the MBB-80. Therefore, the CPM.SYS had to be reconstructed.

The files DKPRM.DEF and CONFIG.DEF were first checked to ensure that the desired hardware configuration

was accurately reflected in the Disk Definition Tables, the Disk Tables, and the Bubble Tables. Once this was completed, the file MBBIOS.A86 was reassembled and was then concatenated with CPM.H86 using PIP.COM. The resulting hex file was then converted to an executable CMD file and renamed CPM.SYS.

To ensure that all possible errors were avoided prior to system initialization, it was decided to reformat the MBB-80. The program MB80FMT.CMD was executed, inserting 8000h as the MBB-80 controller base address. Once formatted, the CP/M loader program MB80LDR.CMD was placed on tracks 0 and 1 of the MBB-80 utilizing the LDCOPY.CMD utility. The reconstructed system was booted and functioned normally.

3. Modification of the BIOS for Use in the iCS-80

As envisioned in the program development process, new programs would be transferred to the MDS double density system using a laboratory utility program. These programs could then be placed on the MBB-80. The MBB-80 would then have to be physically moved to the iCS-80 chassis. By entering the command GFFD4:4, CP/M-86 could be booted and the programs executed under CP/M or DDT86.CMD. However, since the MBB-80 would be the sole memory storage device in the iCS-80 chassis, a new modified BIOS had to be constructed.

The changes that needed to be made were located in two major areas of the BIOS. First, the file DKPRM.DEF which contained the disk definition statements for each logical CP/M disk drive had to changed. The number of logical devices was changed to 1 and the disk definition statement for the MBB-80 was entered as CP/M logical drive 0 (Drive A:) indicating that the MBB-80 was the only "drive" in the system. The other changes were made to the Disk and Bubble Tables contained in the file CONFIG.DEF. Eicklin and Neufeld had created these tables to identify whether CP/M logical drive numbers where either MBB-80 devices or i202 controllers. These tables would support any hardware configuration of MBB-80's and i202 contollers up a total of 16 disk drives (maximum for CP/M). However, other peripheral devices such as the REMEX Data Warehouse could not be supported as was initially believed.

Once these changes had been made, the BIOS was reassembled and used to create a new CPM.SYS which was placed on the bubble memory. It was subsequently tested and it functioned normally.

4. <u>REMEX Low-Level Routines</u>

Concurrently with the work on the MBB-80, low-level read/write routines were written and executed which accessed the REMEX Data Warehouse memory storage unit. This work was accomplished on the iCS-80 chassis using an iSBC 86/12A single board computer and the REMEX Multibus Interface Card

Assembly. At first only the most primitive operations were performed, since there was no permanent memory in the system. Using the 957 Monitor program, small programs were executed to examine the various values contained in the interface status registers. Once the new MBB-80 CPM.SYS was available, more comprehensive programs were written which could build command packets, transmit command packet addresses to the interface board, and check the packet status word for function completion. The basic logic of the read/write functions was discussed in greater detail in Chapter 3 and the logic diagram is shown in Figure 4.1.

A command packet was built which would write a very simple set of characters to a particular head, track, and sector number of the REMEX hard disk or a track and sector number on the floppy diskette. Using DDT86.CMD, the command packet was then altered to produce a read operation which would retrieve the previous message from the RDW and write it to a selected memory address. DDT86.CMD was also extensively used to monitor packet construction and memory content. With each successful transfer, larger blocks of data were transferred until it was concluded that the operations were being correctly performed. Although some progress was made, the program turn-around time resulting from the lack of an adequate development system definitely impeded further progress.

```
                   |--------|
                   | Start  |
                   |--------|
                       |
            |-------------------|
  |-------->|   Check Packet    |
  |         |    Ready Flag     |
  |         |-------------------|
  |                   |
  |                  /\
  |          No     /Is\
  |----------------/ it \
                   \set?/
                    \  /
                     \/
                     |  Yes
            |-------------------|
            |   Build Packet    |
            |-------------------|
                     |
            |-----------------|
            |   Send Packet   |
            |-----------------|
                     |
       |---------------------------|
       |  Check Status Word        |<-----------|
       |---------------------------|            |
                     |                          |
                    /\                          |
                   /Is\          Yes            |
                  / it \-------------------------
                  \ =Ø?/
                   \  /
                    \/
                    |  No
                    /\
                   /Is\       No
                  / it \-----------> Error
                  \ Ø1?/
                   \  /
                    \/
                    |  Yes
                    |
       |-----------------------------|
       |   Operation Complete        |
       |-----------------------------|
```

Figure 4.1
RDW Read/Write Logic

5. Table-Driven BIOS

The development mechanism that had been used up to this point was tedious and time-consuming. The time required to repair errors discovered while working on the iCS-80 was too excessive to support cohesive program modification. It also became evident that the concepts used by Hicklin and Neufeld in the development of their BIOS were not sufficient to meet the objectives of this thesis. Although it was presented as a model for a very flexible system, the BIOS actually only supported MBB-80 bubble memories and 1202 floppy diskette controllers. Inclusion of additional peripheral devices would have required major modification to the BIOS. Furthermore, even if these modifications were made, each time a device was added or deleted from the system, the code within the BIOS for the individual function calls would have had to be changed. The many inconveniences of the program development procedure coupled with the limitations of the Hicklin and Neufeld approach in a varying hardware environment necessitated a new BIOS design strategy.

Hammond [Ref. 3] had identified that certain device specific code could be extracted from the "core" of the BIOS without affecting function operation. This was accomplished by indirectly vectoring BIOS calls to the proper subroutines via a table of labels. Hammond had extracted the READ, WRITE, and INIT BIOS functions and constructed the

appropriate tables in a separate file named CONFIG.DEF. This file was then assembled with the BIOS by means of an "include" statement.

Next, let us examine the READ function in greater detail to see exactly how this BIOS works. Figure 4.2 contains the code for the READ function in the "core" BIOS for a hardware configuration consisting of an 1201 Floppy Disk Controller.

```
read:
        xor  bx,bx
        mov  bl,unit
        add  bx,bx
        call readtbl[bx]
        ret
```

Figure 4.2
Table-Driven BIOS Read Code

This controller supports two floppy disk drives which correspond to CP/M logical drives 0 and 1. This correspondence is set up in the Disk Definition Tables. Also prior to the BDOS call to the BIOS READ function, the desired drive number has been stored in a BIOS variable called "unit". The value of "unit" is first placed in the "bl" register. Next, it is doubled since each label in the read_table represents the 16-bit address of the device-specific read functions. A call is now made to the read_table using the offset contained in the "bx" register. This table entry then indirectly addresses the appropriate subroutine for the desired "unit". For example, if CP/M

logical drive 1 (B:) is selected, the read call is indirectly addressed to the subroutine label located at an offset of two (2) in the read_table. The read_table is shown in Figure 4.3. Notice that since both CP/M logical drives are floppy disk drives, the read call is vectored to the same subroutine.

```
readtbl dw offset i201_read
        dw offset i201_read
```

Figure 4.3
BIOS Read Table

Through the use of a table-driven BIOS, the configuration flexibility needed for this application could be achieved. The use of the indirect call allows all device specific code to be isolated in a single file. Therefore, a separate file can be constructed for each unique peripheral device and can be included in the BIOS by the use of the "include" assembly command. An additional benefit of this type of approach is that it allows for the systematic addition or deletion of hardware devices to or from the system without disturbing the basic BIOS code.

The table-driven concept also provided an improved program develpment scheme and a more logical approach for the implementation of the REMEX Data Warehouse memory unit. Hammond had previously written the code to support the Intel i201 Floppy Disk Controller. A spare i201 controller was

available and was placed in the iCS-80 chassis. With a few minor modifications to the BIOS, it was operational in a very short time. Since both ALTOS and MDS single-density diskettes were fully compatible, programs could now be written, assembled, and converted to executable code and then be taken directly to the iCS-80 for execution. This reduced the amount of time needed to correct errors or modify a program and greatly facilitated code generation.

Because devices could be added to the BIOS independently, it was decided to utilize the 1201 floppy disk drive as a developmental aid and to subsequently implement the REMEX floppy disk first followed by the hard disk. The MBB-80 would be substituted for the 1201 once the REMEX interface was completed. This implementation scheme is explained in more detail in the following sections.

B. INTERFACING THE REMEX DATA WAREHOUSE

1. Floppy Disk Drive

During the testing of the initial REMEX READ/WRITE low-level routines, it was observed that the REMEX would only intermittently complete a packet operation. When it did not complete successfully, the program looped infinitely checking the packet status word (see Figure 4.1) for a value other than a zero, indicating that the REMEX had either completed the operation or that an error had occurred. When multiple packets were sent out on the Multibus, completion

codes were occasionally returned in the command packet status word. When DDT86 was used to trace through the Read routine step by step, the same results were obtained. However, this procedure did verify that command packets were being constructed properly and that the packet address was being transmitted to the Multibus correctly.

Next, a Multibus Monitor Board was used to observe the action on the Multibus and confirmed that all data was correct. This led to speculation that either the interface board was not transmitting the correct information to the REMEX or the REMEX was not processing packets correctly once it received the information from the interface board. However, further hardware testing revealed that both the REMEX and the Interface were functioning normally.

The source of the problem was found more by accident than by design. Documentation [Ref. 8 : p. 2-4] indicated that the Interface Assembly would wait from 0 to 15 host CPU cycles between consecutive DMA operations. The exact number of cycles can be jumper selectable by the DMA Throttle. Therefore, polling the packet status word for a completion code was thought to provide sufficient CPU cycles to allow the process to continue. However, when the wiring diagram of the Interface Card Assembly was examined, it was discovered that the DMA Throttle was controlled by the number of Multibus cycles and not by the number of CPU cycles. Since the Throttle was set to the factory default

position, one additional Multibus cycle was required before the interface board could execute its next DMA operation. Because there was only a single host computer in the system, no additional Multibus accesses were made. This explains why marginal success was obtained by sending multiple packets since this provided the additional Multibus accesses. The DMA Throttle jumper was removed which allowed the Interface Card Assembly to respond immediately with a DMA operation once it acquired control of the Multibus. Subsequent packet operations were successfully completed.

Once the READ/WRITE driver routines had been debugged, the next step in the floppy disk implementation was to incorporate these routines into the table-driven BIOS. A separate "include" file called RXFLOP1.A86 was established to contain the necessary device-specific subroutines. Of the seven BIOS functions that had to be addressed, only the READ and WRITE functions required code in addition to that contained in the basic BIOS routines. Each of the other functions were returned directly to the main BIOS.

The command packet was allocated memory space in the data section of RXFLOP1.A86. However, the packet parameters had to be supplied from the BIOS variables in order to access the file requested by the CP/M file manager. Figure 4.4 depicts the READ packet for the REMEX floppy disk drives used in this implementation.

```
                 Bit Number

          15       8 7  4 3   0
             |----------------|
     word 0  | 10h  | 1  | 1/2 |
             |----------------|
          1  |  Status Word   |
             |----------------|
          2  |     Track      |
             |----------------|
          3  | 00   | Sector  |
             |----------------|
          4  |DMA Buffer Offset|
             |----------------|
          5  |      |   00    |
             |----------------|
          6  | 00       64h   |
             |----------------|
```

Figure 4.4
REMEX Floppy Disk Read Packet

From Chapter 3, recall that word 0 of the command packet is composed of a modifiers section, function code block, and unit id number. The value of 10h in the modifiers section merely indicates that a single packet is being sent and that all automatic error routines are in effect. The function code block (1) specifies a READ operation. Since the REMEX floppy disk drives were chosen to be equivalent to CP/M logical disk drives 1 and 2 (B: and C:) for this implementation, the CP/M drive number and the REMEX unit id for the two floppy disk drives were equivalent. Therefore, the desired CP/M disk number is directly inserted into the packet. The 16 -bit BIOS variable "track" which contains the requested track number is placed into word 2 of the command packet. Word 3 which contains the head and selected sector

number is formed by inserting a zero in the upper byte indicating that the floppy diskette will only be addressable on a single side and placing the BIOS variable "sector" in the lower byte. The 20-bit address of the CP/M DMA buffer which will receive the requested data is computed from the DMA base and offset. The extended address bits (bits 16-19) are entered in the lower byte of word 5. For example, if the local memory of an iSBC 86/12A is configured to respond to Multibus memory segment zero, the extended address bits will be equal to 00h. However, if the local memory were configured to respond to Multibus memory segment 1000, then the extended bits would be 01h. The remaining 16-bit address is placed into word 4 of the command packet.

Word 6 which contains the transfer word count caused the most problems with the floppy disk interface. The major difficulty encountered was the direct result of poor and misleading documentation. The REMEX technical manual for the interface board indicates [Ref. 8 : p. 2-4] that the REMEX can selectively transfer data to the host computer in either 8-bit or 16-bit words by setting a single switch. Since CP/M works with 8-bit words, the switch was set accordingly and a transfer word count of 128 8-bit words was placed in the packet and sent to the REMEX. At first, this seemed to work correctly because a directory of the diskette was read without difficulty and files could be transferred to and from the diskette without error. However, problems

were encountered when attempting to execute a file that was on the diskette. An error message of "FILE NOT FOUND" was displayed intermittently. If a file was found, the program would not execute correctly. In both cases, the system partially crashed and no other operations could be accomplished, despite the fact that the prompt character continued to function normally along with an occasional error message.

The source of the problem was not readily apparent. The operating system worked correctly until the directory of the REMEX floppy diskette was obtained or a file was executed. However, no error code was being generated by the REMEX. In fact, the success code that was being generated indicated that the operation and data transfer was being correctly accomplished. Executing the routines using DDT86 also indicated that the REMEX was functioning correctly and showed that the data was being placed in CP/M DMA buffer.

Numerous changes and experiments were made attempting to locate the cause of this problem. Printouts of the diskette's directory were obtained without error. Hardware was tested and retested with negative results. Finally, a memory map of the operating system was printed after obtaining the directory from a diskette in the MDS single density disk drive system. This was compared to a memory map of the operating system after the directory of

the same diskette was taken from the REMEX floppy drive. It was here that the error was uncovered. The REMEX was transferring 256 8-bit words into the DMA buffer space, not the 128 8-bit words as believed. Thus, the extra data was overwriting portions of the CP/M-86 BIOS causing the system to partially crash. The problem stems from the fact that the REMEX wants to know how many 16-bit words it should transfer. This is completely independent of how the REMEX will transmit the data. Therefore, since a CP/M sector of 128 bytes is equivalent to 64 or 40h 16-bit words, 40h was placed in word 6 of the command packet and no further problems were encountered.

2. Hard Disk

Although the implementation of the hard disk was very similar to the floppy disk drives, there were some notable exceptions. First, the REMEX had a sector size that was a multiple of the standard CP/M sector size of 128 bytes. This necessitated the use of a sector blocking/deblocking routine to resolve this disparity. Second, since the REMEX hard disk has four (4) separate heads, the question of how to divide up the disk had to be resolved. The most logical and straightfoward method was to let each head represent a separate CP/M logical disk drive. Each drive would then be able to address up to 4.5 megabytes of data. With these ideas in mind, the hard disk interface was begun.

Changes had to be made to the Disk Definition and Configuration Tables. In the file CPMMAST.DEF, CP/M logical drive numbers 3, 4, 5, and 6 were added to the table. Each drive number had a disk definition statement that described the physical storage capabilities of a single head of the hard disk. The disk definition variables were determined as presented in Chapter 3. Now, the BIOS would support a total of seven (7) peripheral I/O devices: an 1201 floppy disk drive, two REMEX floppy disk drives, and four REMEX hard disk drives. Later, the MBB-80 bubble memory would be substituted for the 1201 disk drive. Also, additional labels had to be added to the tables in the file CPMMAST.CFG to vector the BIOS function calls to the appropriate subroutines located in the "include" file RXHARD1.A86.

The most difficult obstacle to overcome in this portion of the implementation was to determine the REMEX hard disk sector size. The sector size can be either 128, 256, 512, or 1024 bytes. Initially, attempts were made to reformat the hard disk in accordance with Reference 7. Switches S1 and S2 located on the Formatter II Card Assembly were set to configure the hard disk with a 512 byte sector size. A program was then written which built a command packet to execute the REMEX built-in formatting routine [Ref. 7 : p 3-20]. However, repeated attempts failed to produce a successful format operation. The REMEX also supports a built-in maintenance program that tests the Hard

Disk Format operation. When this program was run, multiple error messages were returned indicating that the format program was inoperative.

Since data had been written to and retrieved from the hard disk during low-level driver testing, it was obvious that the REMEX had been previously formatted. The next step was to determine exactly what format was used. This was not as easy as might be expected. During the power up sequence, the REMEX will check the sector size switches and configure its internal circuitry to process sectors of that size even if the switch postions do not represent the actual format of the hard disk. That is precisely why these switches must match the actual physical sector size in order for read/write operations to work correctly. This fact caused considerable confusion in the interpretation of the error messages obtained by attempting to access the border sectors (104, 67, 39, and 21 for sector sizes of 128, 256, 512, and 1024 bytes respectively). However, it was finally determined that the sector size was 512 bytes.

Since the REMEX sector size was a multiple of the 128-byte CP/M sector size, a sector blocking/deblocking routine was needed to coordinate the access of CP/M sectors with the physical sectors of the hard disk. In this case, there were four (4) CP/M sectors contained on each hard disk sector. On each BIOS call, the CP/M-86 BDOS includes information that can be used to provide effective sector

blocking and deblocking. The sector blocking/deblocking routine used in this implementation is distributed by Digital Research in skeletal form [Ref. 6 : p. 70].

The blocking/deblocking algorithms map all CP/M sector read and write operations through an intermediate buffer called "hstbuf". The size of this buffer is equivalent to the REMEX sector size (512). During a read operation, a 512-byte sector of data is read into the "hstbuf" or host buffer from the REMEX hard disk. Since the host buffer now contains four CP/M sectors, the desired 128-byte sector is obtained by correctly offsetting into the host buffer. This data is then transferred to the CP/M DMA buffer defined by the DMA base and DMA offset variables. Similarly, during a write operation, four CP/M sectors are written to the host buffer. The data is then transferred to the REMEX hard disk and stored on a single 512-byte sector.

Within the blocking/deblocking routine itself, the values and variables which relate to CP/M sectors are prefixed by "sek", while those related to the REMEX hard disk are prefixed by "hst". The SELDSK, SETTRK, SETSEC, SECTRAN, and SETDMA entry point routines were transposed into the REMEX hard disk "include" file. These subroutines store values for later use and SECTRAN translates CP/M sector values into the corresponding physical sector. The READ and WRITE entry point labels were placed in the read_table and write_table respectively, while the actual

REMEX hard disk read and write low-level drivers were incorporated at the READHST and WRITEHST entry points.

The command packet was constructed from the following variables: "hstdsk" which represents the host disk number, "hsttrk" which is the host track number, and "hstsec" which cooresponds to the host sector. The host disk number is transformed into the appropriate head number and is entered into the upper byte of word 3 of the command packet. The memory segment and offset of the host buffer (hstbuf) is translated into a 20-bit address. The extended bits (16-19) are entered into the lower byte of word 5, while the remaining 16-bit address is placed in word 4 of the command packet. For the REMEX hard disk, we want to transfer 512 bytes or 256 16-bit words. Therefore, the

Bit Number

	15 … 8	7 … 4	3 … 0
word 0	10h	1	0
1	Status Word		
2	hsttrk		
3	head #	hstsec	
4	16-bit addr of hstbuf		
5		ext bits	
6	100h		

Figure 4.5
REMEX Hard Disk Read Packet

transfer word count (word 6) was set to 100h. The REMEX hard disk Read packet is shown in Figure 4.5.

3. Initial Multi-iSBC 86/12A System

The above implementation produced a CP/M-86 BIOS that supported the MBB-80 bubble memory and the REMEX Data Warehouse floppy and hard disk drives. The original master iSBC 86/12A (#1) was booted from the MBB-80 and had its onboard memory switch-and-jumper selected to be accessible from the Mulitibus beginning at memory segment zero. Data transferred from the REMEX would be put directly into the CP/M DMA or Host Buffers via DMA operations. The next step was to introduce a second iSBC 86/12A into the system which would also utilize the CP/M-86 operating system.

It was decided to use the 32K common memory to hold a bootloader program that could be used by the slave iSBC 86/12A computers to boot the CP/M-86 system. A utility program, LDCPM.A86, was written to place a copy of CP/M-86 into common memory which was especially configured for the slave computers. A second utility, LDBOOT.A86, was used to transfer a copy of the bootloader program (BOOT.A86) into common memory. The resulting common memory map is shown in Figure 4.6. CPMSLAVE.CMD was identical to the CP/M-86 system used for the master iSBC 86/12A except that it supported an iSBC 86/12A whose local memory was accessible from the Multibus beginning at memory segment 1000h. When initiated from the iSBC 86/12A monitor, the bootloader program would

```
E000:0000  |--------------|
           |  BOOT.CMD    |
           |--------------|
           |              |
     :1000 |--------------|
           | CPMSLAVE.CMD |
           |              |
     :4000 |--------------|
           |    FREE      |
           |             /
           |           /
           |         /
           |       /
           |     /
```

Figure 4.6
Common Memory Map

transfer the CP/M-86 slave system from common memory into local memory beginning at 40:0000h. Once the transfer was complete, control would be passed to the BIOS to initialize the system. *It must noted that all these programs must reside on CP/M logical drive D:.*

This scheme, although utilizing the DMA capability of the REMEX to the maximum extent possible, would require a different CPMSLAVE.CMD file for each iSBC 86/12A added to the system. Each computer's local memory would have to be placed in a separate 64K block within the one-megabyte address space available to the Multibus and these page numbers would be have to be entered in the lower byte of word 5 in the command packet. This organization is somewhat awkward and exhausts a large portion of common memory if

several computers are used. Therefore, a more acceptable alternative was needed.

C. SYNCHRONIZATION AND PROTECTION

1. Synchronization of Read/Write Operations

With two active iSBC 86/12A computers in the system, the synchronization of read/write operations had to be addressed. Since the REMEX could queue up to eight (8) command packets internally, it was initially felt that this feature would provide adequate synchronization of the I/O requests from the independently operating computers. However, when simultaneous multiple transfers were attempted between the CP/M hard disk logical drives, sporadic errors occurred. Inspection of the READ and WRITE routines in the hard disk "include" file (RXHARD1.A86) revealed that there was nothing to prevent a clash of both iSBC 86/12A computers if they simultaneously attempted to send a command packet address to the REMEX Interface Card Assembly. Since the packet addresses were sent in three (3) single-byte Multibus transfers, it was indeed possible for the values sent to the interface board to become intermixed. Also, once the most significant byte of the packet address is sent, the interface immediately signals the REMEX that the packet address is complete and ready to be transferred. However, this may not be the case. Consider the case where computer #1 has transferred the extended address and the least significant bytes of the packet address to the Packet/DMA

Register. Computer #2 then sends the extended bits of its packet address. Since each computer's memory begins on a different page, the extended bits will be different for each iSBC 86/12A. Computer #1 now regains control of the Multibus and sends its most significant address byte. The Remex will now read the packet located in computer #2's address space rather than the packet in computer #1's address space. This will certainly cause severe problems.

Initially, the section of code used to send out the packet address was identified as a critical section. A semaphore was then defined to control the access to the critical section. In order that all active iSBC 86/12A computers could have access to the semaphore, it was placed in common memory and could take on a value of either Ø or 1 indicating that the resource was either busy or free respectively. If it was a 1, the requesting computer would set it to Ø, send the three bytes of the packet address, and then reset it to 1. If the requestor found that the semaphore was equal to Ø, it would delay and then recheck. This checking process was implemented using the LOCK XCHG instruction to provide exclusive use of the Multibus.

When simultaneous multiple file transfers were again attempted, errors still occurred indicating that there was still some interference on the Multibus. This probably occured when the registers of the interface were set up for a DMA data transfer and a packet address was then written

into the Packet/DMA register before the data could be transfered. At any rate, a more inclusive synchronization scheme was required to ensure that a single iSBC 86/12A read/write operation could be completed without encountering contention from the other computers in the system.

Since it was desirable to have all iSBC 86/12A computers configured alike, it was decided to adopt a software approach to the synchronization problem rather than the conventional monitor approach. The method chosen was based on sequencers and eventcounts [Ref. 12]. This method is modeled after the "ticket/server" system used in many stores where services are performed. When the customer arrives, he takes a numbered ticket and then waits for his number to come up before being served. The server works in ticket number order. The implementation of this scheme is very straightfoward and had been previously used by Hammond [Ref. 3]. Two 16-bit counter variables, "ticket" and "server", were placed in common memory. The value 0 was reserved for the ticket number indicating that another computer was presently modifying the ticket number. Exclusive access to the ticket number was provided by the LOCK XCHG instruction. An algorithmic language representation of the sequencer routine is given in Figure 4.7. The delay used in the Await Subroutines was used to prevent Multibus contention. "Request" is called prior to each read or write operation to gain exclusive access to the

```
*************************************************************
                   Primitive Subroutines
*************************************************************

        ticket:                 ;return a ticket number

                  customer no. = ticket no.
                  inc ticket no.
                  ret

-------------------------------------------------------------
        await:                  ;delay until customer no. =
                                ;server number

                   while customer no. < server
                         delay
                   ret

-------------------------------------------------------------
        advance:                ;inc server

                   inc server
                   ret

*************************************************************
                   Entry Point Routines
*************************************************************

        request:                ;get resource

                   call ticket
                   call await
                   ret

-------------------------------------------------------------
        release:                ;release resource

                   call advance
                   ret
```

Figure 4.7
Sequencer Algorithm

shared resource. Once the operation is complete, "release" is called to free the resource by incrementing the server number which allows the next I/O function to be executed. When the sequencer code was implemented into the read/write routines for each of the peripheral I/O devices, no further errors were noted.

2. Common Memory Read/Write Routines

As alluded to earlier, the CP/M-86 BIOS which uses DMA operations to transfer data between the iSBC 86/12A computers and the Remex Data Warehouse requires a unique BIOS for each computer in the system. This places a severe limitation on further system expansion and complicates the system configuration control requirements. Futhermore, this type of implementation requires that at least a portion of the iSBC 86/12A's local memory be accessible to the Multibus. One of the principal goals of this thesis was to provide a system in which all computers were isolated from one another. Obviously, this implementation does not support this goal. It also results in an awkward bootloader arrangement in common memory and requires that all versions of CP/M-86 needed for system operation be accurately updated should any changes or modifications occur. Therefore, a more acceptable BIOS implementation had to be found.

The resulting implementation routed all data transfers through a common memory buffer. The size of this buffer was set to correspond to the largest physical sector

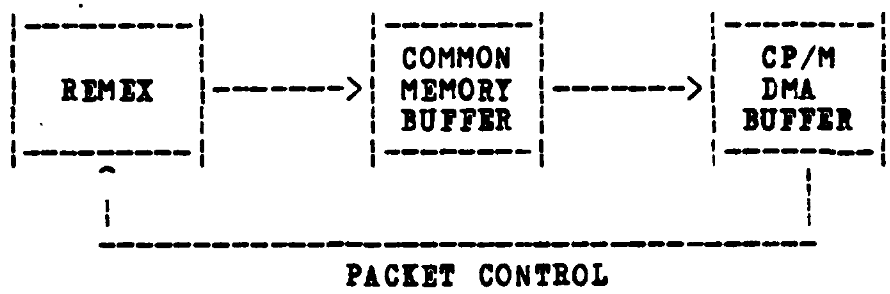


Figure 4.8
Common Memory Read Operation

within the system which was the 512-byte sector of the REMEX hard disk. The additional code required for each read/write routine was minimal since its only function was to transfer a given amount of data between local and common memory. A data flow diagram depicting a typical read operation from the REMEX is shown in Figure 4.8. For illustration, consider a CP/M initiated read operation from the REMEX hard disk. The command packet will be constructed as before except that the 20-bit common memory buffer address will replace the host buffer address in word 4 and the lower byte of word 5 of the packet. This will result in the desired data being read into the common memory buffer. When this operation is complete, the requesting iSBC 86/12A will then transfer the data in the common memory buffer to the host buffer located in the data section of the CP/M BIOS. This procedure is entirely transparent to the CP/M BDOS. A write operation is similarly completed. First, the data in the host buffer is written to the common memory buffer. Next, a packet is sent

to the interface which transfers the data from common memory to a specified head, track, and sector of the hard disk. The required changes were made to the "include" files for the MBB-80 bubble memory, the REMEX floppy disk drives, and the REMEX hard disk drives and the files were renamed MB80DSK.A86, RXFLOP.A86, and RXHARD.A86 respectively. These files appear in Appendices C, D, and E.

The common memory routines produced several improvements to the overall system design. First, all iSBC 86/12A computers could be completely isolated. Each of the four computers used in the system was jumper configured so that all onboard memory was reserved totally for local CPU use and could not be accessed from the Multibus. This provided the required protection for each computer's local

```
E000:0000  |---------------------|
           |ticket|server|       |
           |---------------------|
    :0100  |---------------------|
           |     CP/M Buffer     |
    :0300  |---------------------|
    :0400  |---------------------|
           |      BOOT.CMD       |
    :0500  |---------------------|
           |                     |
           |    CPMSLAVE.CMD     |
           |                     |
    :3500  |---------------------|
           |                     |
           |        FREE         |
           |                     |
```

Figure 4.9
Common Memory Allocation

memory. Second, only a single copy of the CP/M-86 operating system was required for all of the slave computers since data transfers were locally initiated. In fact, the only difference between the slave and master versions involved the initialization of the synchronization variables and the log-table. A memory map showing the configuration of common memory is presented in Figure 4.9.

3. Disk Write Protection

The ticket/server synchronization routine ensures that single iSBC 86/12A read/write operations can be completed without interference. However, this is not sufficient to provide the necessary write protection to the shared devices in a system of multiple computers each running CP/M-86. Consider the case of two processors trying to write to the same CP/M logical disk. CP/M reads the disk directory and constructs an allocation vector in the BIOS that indicates the logical blocks on the disk that have not been written to previously. Each iSBC 86/12A then proceeds to write its data file to the unallocated blocks in sequential order. Although the individual write operations were synchronized, the result is still overwritten and garbled data. Therefore, this implementation institutes a read/write strategy that allows all computers to read data from all the shared devices but only write to a single device to prevent files from being overwritten. Moreover, it was also desirable to be able to select any of the shared

devices for write operations from each of the four system console positions.

A logon and logout procedure was developed to control the write access to the various peripheral devices through the use of a table located in common memory. This table has a entry for each device in the system (Figures 4.10 and 4.11). Before the user is permitted to boot the CP/M he is asked for his console number and the CP/M drive that he wishes to log onto (write to). The CP/M drive

	A:	B:	C:	D:	E:	F:	G:
logtbl	MBB-80	FLOP1	FLOP2	HARD1	HARD2	HARD3	HARD4

Figure 4.10
Login Table

```
E000:0000  |-------------------------------|
           | ticket | server |    logtbl    |
           |-------------------------------|
     :0100 |-------------------------------|
           |          CP/M Buffer          |
     :0300 |-------------------------------|
           |                               |
```

Figure 4.11
Final Common Memory Configuration

number is stored in a local variable called "user" which is used as an offset into the log table. The log table is then checked to determine if the desired disk has already been logged onto. If not, the console number is entered into the

log table at an offset corresponding to the given device. Otherwise, the user is asked to select another disk. To log out, the user types the command "logout" which places a zero (free) in the log table at an offset equal to the user number. Each CP/M logical disk drive requires its own copy of the log out routine (LOGOUT.CMD) so that it can be executed from every disk drive.

Within the BIOS, when a write operation is requested, the variable "user" is compared to the CP/M logical disk number. If they are equal, the write operation is permitted to continue. If not, the user is informed that write operations are not permitted to that disk drive. This guarantees that no two iSBC 86/12A computers can write to the same shared disk.

D. SUMMARY OF SYSTEM GENERATION

The following descriptions provide step-by-step procedures on how to create the BIOS for this implementation of the CP/M-86 operating system, how to step up the MBB-80 bubble memory board in the MDS double density system, and how to start up the multi-user CP/M-86 system.

1. System Bios Creation

a. Develop separate files for each I/O device being sure to address the seven device specific functions in each. In this code, before any Multibus access include the command "call request" and upon completion of a Multibus access include the command "call release".

b. Ensure that all I/O is accomplished via the common memory I/O buffer which extends from E000:100 to E000:300. Develop a transfer routine for moving data to and from the common memory buffer and the host computer.

c. Decide upon the logical hardware configuration as will be seen by CP/M-86. Based on this configuration, develop the Disk Parameter Table which will be used as the source file for GENDEF.CMD to produce a ".LIB" file. Also, using this same hardware configuration and the I/O device files, develop the label tables in CPMMAST.CFG for the seven device specific functions.

d. In the BIOS use the "include" command for all I/O device files, the label table (CPMMAST.CFG), and the Disk Parameter Table (CPMMAST.LIB). The files SYNC.A86 and LOGIN.A86 must also be included, but require no modifications.

e. Assemble the BIOS using ASM86.COM. Using ASM86.CMD may generate forward reference errors and require the rearrangement of some included files in the BIOS. Two assemblies must be made. The first must be assembled with the master conditional assembly switch set to true in order to create the master BIOS. The second must be made with the switch set to false in order to create the slave BIOS.

f. Concatenate the resulting hex files with CPM.H86 to form CPMMAST.H86 and CPMSLAVE.H86. Use the CP/M utility

command GENCMD.CMD (GENCMD CPMMAST 8080 code[a40]) to generate the executable command files.

g. Transfer CPMMAST.CMD to the MBB-80 bubble memory board as CPM.SYS. Transfer CPMSLAVE.CMD to drive D: of the REMEX.

2. <u>Setting up the MBB-80 in the MDS System</u>

a. Remove the Intel 8080 microcomputer and the associated memory boards from the MDS double density system.

b. On the iSBC 86/12A #1, place the switches 1-16 and 8-9 on DIP switch S1 in the closed position. Install a jumper between pins 127 and 128. If there are jumpers in place for the clock, pins 103 and 105, remove them.

c. Insert the iSBC 86/12A #1 and the MBB-80 board with the backplane into the MDS chassis.

d. Turn the power to the MDS chassis and the disk drives on. Once these devices are running, apply power to the MBB-80 board by setting the memory protect switch on the backplane to the "run" position. Now, the CP/M-86 operating system can be booted from a double density diskette by entering the command GFFD4:0. The system booted should be one that is capable of addressing the bubble memory as a diskette.

e. To format the MBB-80 bubble memory execute the program MB80FMT.CMD and use 8000H as the base address for the controller. Execute LDCOPY.CMD using LDRMB80.CMD as the source file. This will place the loader on tracks 0 and 1

of the MBB-80 bubble board. Finally transfer CPMMAST.CMD to the bubble as CPM.SYS.

3. System Initialization

a. Insert four iSBC 86/12A computers into the iCS-80 chassis. One computer must have a jumper on pins 103/104 and 105/106. These connections supply the clock for the Multibus. All computers should have pins 112 and 114 connected by a jumper wire. This ensures that the computer's local memory is inaccessable to the Multibus. Also on all computers, only position 8-9 on DIP switch S1 should be closed. All other positions should be open. Finally, insert the MBB-80 bubble memory board, the 32K common memory board and the REMEX interface board into the iCS-80 chassis.

b. Turn the iCS-80 power switch on.

c. Power up the REMEX in accordance with Ref. 7 and turn the MBB-80 memory protect switch to "on". This switch is located in the rear of the iCS-80 chassis.

d. When the REMEX hard disk has timed out and the ready light is on, enter the command GFFD4:4 from the console attached to iSBC 86/12A #1 to boot CP/M-86 from the MBB-80. The synchronization variables and the log table entries will be initialized in common memory.

e. Select drive D:

f. Execute LDCPM located on drive D:. This will load the file CPMSLAVE.CMD into common memory starting at E000:500.

g. Execute LDBOOT located on drive D:. This will place the file BOOT.CMD into common memory starting at E000:400.

h. Now, CP/M-86 can be booted on any iSBC 86/12A computer by entering the command GE000:0400 from the monitor.

i. When a session is completed, enter the command LOGOUT to logoff the system.

V. RESULTS AND CONCLUSIONS

A. GENERAL RESULTS

The ultimate goal of this thesis was to develop a multi-computer "protected" CP/M-86-based system that shared memory storage devices. This goal was accomplished and the resulting code is located in the Appendices. The major product produced by this thesis is a completely operational multi-user development station. The CP/M BIOS is completely table-driven and can be reconfigured for different hardware configurations in under twenty minutes. This feature alone is a significant improvement over the standard BIOS marketed by Digital Research. In addition, it should be quite easy to expand the current system to permit more users or add additional I/O devices.

The system provides user protection in several forms. No user, once logged onto the system can destroy, either by design or by accident, anothers user's files or local CPU memory. However, any single computer can destroy common memory, but it is a simple matter to restore it. Furthermore, the logon and logout procedures prevent two users from simultaneously logging onto and writing to the same CP/M logical disk drive.

B. EVALUATION OF THE IMPLEMENTATION

To evaluate system performance, two tests were conducted. The first test involved assembling a 3K and then a 24K file with a single computer logged onto the system. The assembly time was recorded using a conventional stopwatch. Next, two computers were used to simultaneously assemble the same file, followed by three and then four computers. The results of the test are shown in Table 5.1.

Table 5.1
REMEX Assembly Times In Seconds

FILE SIZE	ONE COMPUTER	TWO COMPUTERS	THREE COMPUTERS	FOUR COMPUTERS
3K	12.9	22.1	25.1	28.6
24K	211.1	246.7	257.3	275.5

Table 5.2
MP/M Assembly Times In Seconds

FILE SIZE	ONE USER	TWO USERS	THREE USERS	FOUR USERS
3K	22.3	X	X	X
24K	323.2	X	X	X

One might expect that two computers would take twice as long to assemble the same program and three computers three times as long. However, except for the initial contention for the I/O devices, all computers could assemble the files in parallel. This accounts for the fact that there is not a

linear relationship between the number of computers operating in the system and the assembly times.

To provide a means of comparison, an attempt was made to run the same test under the MP/M operating system. However, MP/M would not permit more than one file to be assembled at the same time. In fact, on several attempts, the entire system crashed. The results of this test are shown in Table 5.2.

The second test involved a file transfer utilizing the CP/M-86 utility PIP.CMD. Since all operations were I/O intensive, this test represented a worse case scenario. The first run consisted of transferring a 16K file with only one computer operating in the system and recording the time it took to complete the operation. Then two and finally three computers were used to execute the identical PIP command at the same instant. The time it took for all computers to complete the task was recorded. The results of these tests are shown in Table 5.3. The "Xs" indicate that it was not possible to make the transfer because there was an insufficient number of destination type devices. (i.e. Two computers cannot transfer files to a single bubble device at at the same time.)

To provide a comparison for the above results, the same test was run on the MP/M system. Although the two system configurations are different, they do offer some basis for comparison. However, in the MP/M system, only operations

Table 5.3
REMEX Transfer Times In Seconds

FROM \ TO	HARD DISK	BUBBLE DEVICE	FLOPPY DISK
SINGLE COMPUTER EXECUTING PIP			
HARD DISK	2.5	5.6	8.1
BUBBLE DEVICE	5.6	8.0	11.6
FLOPPY DISK	7.3	9.6	12.0
TWO COMPUTERS EXECUTING PIP			
HARD DISK	5.9	X	54.4
BUBBLE DEVICE	11.3	X	54.6
FLOPPY DISK	29.1	X	X
THREE COMPUTERS EXECUTING PIP			
HARD DISK	10.6	X	X
BUBBLE DEVICE	18.4	X	X
FLOPPY DISK	49.7	X	X

between the hard disk and floppy disk were possible. The results of this test are shown in Table 5.4.

From these results, it can be seen that the multi-user CP/M-86 system has a slight performance advantage for single user disk operations. When more than one user is operating in the system, this performance advantage becomes very

Table 5.4
MP/M Transfer Times In Seconds

FROM \ TO	HARD DISK	FLOPPY DISK
ONE USER EXECUTING PIP UNDER MP/M		
HARD DISK	7.3	12.0
FLOPPY DISK	11.2	14.8
TWO USERS EXECUTING PIP UNDER MP/M		
HARD DISK	17.4	26.2
FLOPPY DISK	26.3	X
THREE USERS EXECUTING PIP UNDER MP/M		
HARD DISK	23.7	X
FLOPPY DISK	36.9	X

significant for transfers made between areas on the hard disk. However, the REMEX floppy disk drives are slower.

Since the REMEX hard disk can be used to emulate the "signal processor" functions of the AEGIS system, a third test was conducted to determine the optimum skew factor for consecutive read operations. A low-level routine was written to continuously read sectors from the hard disk into common memory. After each read operation, a counter was incremented. When five read operations had been completed, a character was printed to the CRT screen. The time it took to print 80 characters to the CRT is recorded and

Table 5.5
REMEX Winchester Disk Skew Times
in Seconds

SKEW FACTOR	TOTAL TIME	SKEW FACTOR	TOTAL TIME
0	10.00	20	5.25
1	10.35	21	5.55
2	10.55	22	5.80
3	10.95	23	6.10
4	11.25	24	6.35
5	11.45	25	6.60
6	11.70	26	6.85
7	11.95	27	7.10
8	12.20	28	7.35
9	12.55	29	7.55
10	12.75	30	7.80
11	13.05	31	8.05
12	13.40	32	8.30
13	13.45	33	8.65
14	13.70	34	8.85
15	4.20	35	9.20
16	4.35	36	9.45
17	4.55	37	9.65
18	4.85	38	9.85
19	5.05		

approximates the time it took to conduct 400 separate read operations. During the first run, the skew factor was set to zero. Therefore, no sectors were skipped between read operations. In the subsequent runs, the skew factor was incremented by one for each successive test. The results are shown in Table 5.5 and indicate that a skew factor of 15 is optimal for reading data from the REMEX hard disk.

C. RECOMMENDATIONS FOR FUTURE WORK

There are several possible opportunities for future projects involving the REMEX hard disk and the multi-user

CP/M-86 system. The first and foremost is the use of the system to emulate the AEGIS system. Several AEGIS system modules have already been developed and could be run on dedicated iSBC 86/12A computers using the REMEX hard disk to supply simulated radar data. In the present hardware configuration, four system modules could be run concurrently.

However, there are other smaller support projects which would increase the capability and utility of the system. There is an urgent need for a more sophisticated text editor or word processor. Without one, the system will not be used to its full capabilities. Translating the 8080 assembly language code of BTED.COM into 8086 assembly language would provide a more usable text editor than the one currently provided by Digital Research - ED.CMD.

Another possible project is to develop a boot loader program for the REMEX Data Warehouse. As the system is currently designed, the CP/M operating system must be initially loaded from either the MBB-80 or from the MDS single density system. This would allow CP/M to be booted from any of the memory storage devices currently in the system.

A more ambitious project would be to design a boot loader which permitted the user to boot not only the master CP/M-86 operating system directly from the REMEX Data Warehouse, but the slave CP/M-86 operating system as well. This would relieve the master system of the task of loading

the CP/M slave system and the boot loader program into common memory prior to booting the other slave computers. Furthermore, it would free a larger portion of common memory for general use and decrease the number of system variables that would have to be reconstructed should common memory be destroyed. The programs LDCPM.A86, LDBOOT.A86 and BOOT.A86 which are already written could be combined to form the nucleus for such a program. Once operating correctly, the program would have to be loaded into an iSBC 86/12A EPROM where it would be accessible to the monitor.

The final project could alter the CP/M-86 BIOS to include the Micropolis Winchester hard disk, the MDS double density disk drive system, and the newly acquired 256K bubble memories. The code for the Micropolis hard disk and the MDS double density disk drive system has already been written and only needs to be put into the table-driven BIOS format. The implementation of the new bubble memories should be very similar to that of the MBB-80.

APPENDIX A
PROGRAM DISCRIPTIONS

I. MBB-80 BUBBLE MEMORY FILES

A. MB80FMT.CMD: This program is used to initially format the MBB-80 bubble storage device as a single density disk drive. When the program is executed it will prompt the user for a segment address. The address of 8000 must be entered. The program will then set the controller base address to 8000h and write the correct byte patterns on the bubble memory system to give it the appearance of a diskette. [Ref. 2 : p. 88 and p. 159]

B. MB80ROM.A86: This file contains the source code necessary for bootstrapping the system from the bubble memory device. It has been loaded into an EPROM and placed on the motherboard of the iSCB 86/12A computer labeled #1. It is executed by entering the command GFFD4:4 into the monitor of the computer. The program will then place the system loader into memory and transfer control to it. [Ref. 2 : p. 187]

C. LDRMB80.CMD: This is the loader program that must be placed on the bubble's tracks 0 and 1. It will locate the file CPM.SYS on the bubble memory device, load it into memory and then transfer control to the operating system. The BIOS for this program is created using MBBIOS.A86 with the loader conditional assembly switch set to true.

D. MBBIOS.A86: This file contains the source code used to create the BIOS for both the CPM.SYS and the LDRMB80.CMD The CP/M.SYS BIOS is created with the loader conditional assembly switch set to false. [Ref. 2 : p. 166]

E. DKPRM.DEF: This file contains the hardware configuration tables for arranging up to 16 MBB-80 bubble memory devices or Intel MDS double density disk drive systems in any combination. It was used by Hicklin and Neufeld in their implementation of a "table driven" BIOS. However, different I/O devices (i.e. REMEX Data Warehouse) may not be added to their table. [Ref. 2 : p.95]

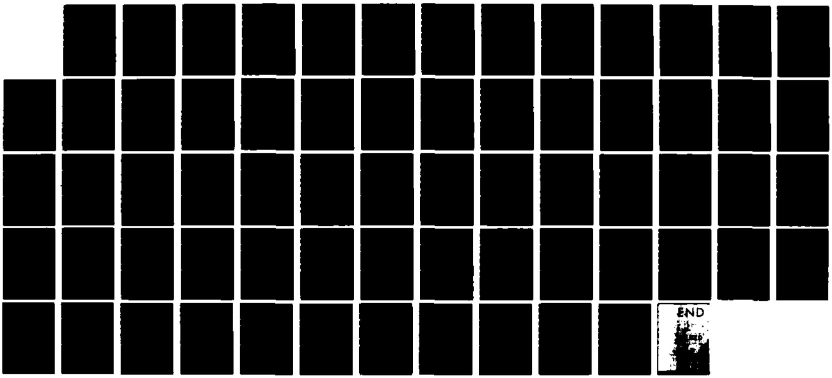
F. CONFIG.DEF: Contained in this file are the disk definition statements used by Hicklin and Neufield to generate the Disk Definition Tables for their BIOS. The file generated is labeled CONFIG.LIB and is included into MBBIOS.A86 when assembled. [Ref. 2 : p. 92] and [Ref. 6 : p. 67]

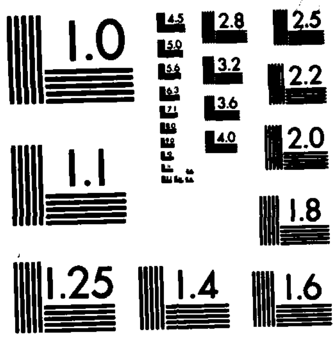
II. REMEX DATA WAREHOUSE FILES

A. CPMBIOS.A86: This file is the basic table driven BIOS used in this thesis. By setting the MASTER/SLAVE conditional assembly switch to either true or false, two different CPM.SYS's can be created. The only difference in the two is that the CPMMAST.CMD system contains code to initialize the synchronization and login variables located in common memory. The resulting MASTER file should be renamed to CPM.SYS and placed on the bubble memory storage

AD-A126 850 ALTERATION AND IMPLEMENTATION OF THE CP/M-86 OPERATING SYSTEM FOR A MULTI-USER ENVIRONMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA T V ALMQUIST ET AL. DEC 82 2/2

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

device. Entering the comand GFFD4:4 from the iSBC 86/12A computer labeled #1 will boot the system.

When the MASTER/SLAVE conditional assembly switch is set to false, a slave system will be created. This system should be named CPMSLAVE.CMD. It is this file that is eventually loaded into common memory via the command LDCPM.CMD.

After the slave system has been loaded into common memory, the command LDBOOT.CMD must also be executed in order to place the loader program into common memory. Once these two commands have been executed, all other computers can issue the command GE000:400 to the computer monitor and the CP/M operating system will be loaded for each.

B. CPMMAST.CFG: This file contains the label tables for the seven I/O device-specific functions which are extracted out of the BIOS. These functions are INIT, SELDSK, HOME, SELTRK, SELSEC, SETDMA, and SETDMAB. A conditional assembly switch is located in the INIT table. When the master switch is set to true, two extra labels are included which permit the initialization of the synchronization and login variables in common memory.

C. MB80DSK.A86: Located in this file is the code necessary to read and write to the MBB-80 Bubbl-Board. It is assembled into the CPMBIOS.A86 file by an "include" statemeмt.

D. RXFLOP.A86: This file contains the code for reading and writing to the REMEX Data Warehouse's two floppy disk drives. It is assembled into the CPMBIOS.A86 file by the use of an "include" statement. Command packets for the REMEX are built in common memory and all DMA is accomplished through common memory.

The file labeled RXFLOP1.A86 is almost identical to RXFLOP.A86. The difference is that common memory is not used for DMA or packet building. Instead the REMEX directly accesses the host's on board memory. Thus RXFLOP1.A86 will only work for a computer which has its local memory address space between 00000h and 0FFFFh. To permit additional computers to use this code, the packet addresses built in this BIOS will have to be changed to correspond to the computer's memory address space within the system's addressable memory space of 1 Megabyte.

E. RXHARD.A86: This file contains the code necessary to access the Remex Data Warehouse's Winchester hard disk. It also contains the blocking and deblocking code required for mapping the REMEX's 512 byte sectors to CP/M's 128 byte logical sectors. It is assembled into the CPMBIOS.A86 file by an "include" statement. Command packets for the REMEX are built in common memory and all DMA is accomplished through common memory.

The file RXHARD1.A86 is almost identical to RXHARD.A86. The difference being that common memory is not used for DMA

or packet building. See RXFLOP.A86 for more detail, as changing RXHARD1.A86 to accomodate more than one user requires the same changes as RXFLOP1.A86.

F. CPMMAST.DEF: This file contains the CP/M-86 disk definition statments used in this thesis. It is the source file for GENDEF.CMD which produces the file CPMMAST.LIB.

G. CPMMAST.LIB: This file is assembled into the CPMBIOS.A86 via an "include" statement. It contains the Disk Parameter Tables created by the CP/M utility program GENDEF.CMD, using the file CPMMAST.DEF as the input file.

H. INTELDSK.A86: While this file is not included in the final hardware implementation of this thesis, it contains the code necessary for accessing the Intel MDS single density disk drive system. It was used extensively in the early developmental phases of this thesis because it provided an easy method of booting a new CPM.SYS. If this file is included into the CPMBIOS, the CP/M-86 operating system can be booted by issuing the command GFFD4:0 to the monitor.

I. LDCPM.A86: This program must be executed in order to load CPMSLAVE.CMD into common memory beginning at E000:500.

J. LDBOOT.A86: This program must be run before the slave CP/M system can be loaded by the other computers. When executed, the program BOOT.CMD will be placed in common memory beginning at E000:400.

K. BOOT.CMD: This is the loader program used by all but the initial computer to boot the CPMSLAVE operating system from common memory. It is executed by entering the command GE000:400 from the monitor after the programs LDCPM.CMD and LDBOOT.CMD have been run.

L. RXFORMAT.A86: When an I/O device is first initialized for use under the CP/M operating system, the hex code E5's must be written on the tracks which will contain the directory, otherwise the error "NO DIRECTORY SPACE" will occur. This program will write E5's on the necessary tracks for each head of the Winchester hard disk. Since executing this program will erase all files accessible to the different heads, it will prompt the user for permission to procede in order to insure that the files are not erased by mistake. Normally this program will not be of any use unless a new hard disk is installed or a directory track is inadvertently destroyed.

M. RXMAINT.A86: The REMEX Data Warehouse contains numerous built-in error checking and maintenance programs which can be implemented by building and then sending maintenance packets to the REMEX. This program prompts the user to choose one of these built-in maintenance programs and then runs the test. If an error is encountered, the error code is printed. The meanings of the error codes can be found in the REMEX tecnical manual. [Ref. 9 : p. 3-19]

N. LOGIN.A86: This file contains the code necessary to

provide protection from more than one user logging on to the same area of the hard disk or the MBB-80 board at the same time.

O. SYNC.A86: This file must be included in the BIOS when more than one computer is going to operate on the Multibus. It contains the code which prevents more than one computer from accessing shared resources while another is conducting a read or write operation through common memory.

APPENDIX B
PROGRAM LISTING OF CPMBIOS.A86

```
;Prog Name   : CPMBIOS.A86 (Master/Slave CPM Bios)
;Modified    : Inclusion of Synchronization Routine
;Date        : 7 October 1982
;Written by  : Tom V. Almquist and David S. Stevens
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Kodres
;Purpose     : This BIOS is for use with the iSB86/12A.
;            : It requires a separate "include" file for
;            : each different I/O device.

;*************************************************************
;                          EQUATES
;*************************************************************

true            equ -1
false           equ not true
cr              equ 0dh         ;carriage return
lf              equ 0ah         ;line feed
error           equ 0ffh        ;general error indication
master          equ true        ;set for master/slave BIOS

;system addresses

bdos_int        equ 224         ;reserved BDOS interrupt
ccp_offset      equ 0000h       ;start of CCP code
bdos_offset     equ 0B06h       ;BDOS entry point
bios_offset     equ 2500h       ;start of BIOS code

;console via the i8251 USART

cstat           equ 0dah        ;status port
cdata           equ 0d8h        ;data port
tbemsk          equ 1           ;transmit buffer empty
rdamsk          equ 2           ;receive data available

        cseg
        org     ccpoffset
ccp:
        org     bios_offset

;*************************************************************
bios:           ;JUMP VECTORS
;*************************************************************

        jmp INIT        ;Enter from BOOT ROM or LOADER
```

```
        jmp WBOOT         ;Arrive here from BDOS call 0
        jmp CONST         ;return console keyboard status
        jmp CONIN         ;return console keyboard char
        jmp CONOUT        ;write char to console device
        jmp LISTOUT       ;write character to list device
        jmp PUNCH         ;write character to punch device
        jmp READER        ;return char from reader device
        jmp HOME          ;move to trk 00 on sel drive
        jmp SELDSK        ;select disk for next rd/write
        jmp SETTRK        ;set track for next rd/write
        jmp SETSEC        ;set sector for next rd/write
        jmp SETDMA        ;set offset for user buff (DMA)
        jmp READ          ;read a 128 byte sector
        jmp WRITE         ;write a 128 byte sector
        jmp LISTST        ;return list status
        jmp SECTRAN       ;xlate logical->physical sector
        jmp SETDMAB       ;set seg base for buff (DMA)
        jmp GETSEGT       ;return offset of Mem Desc Table
        jmp GETIOBF       ;return I/O map byte (iobyte)
        jmp SETIOBF       ;set I/O map byte (iobyte)


;************************************************************
;                  Entry Point Routines
;************************************************************

        include login.a86       ;necessary for multi-users

;------------------------------------------------------------
INIT:   ;print signon message and initialize hardware
        ;and software

        mov  ax,cs              ;we entered with a JMPF
        mov  ss,ax              ;so use cs: as initial
        mov  ds,ax              ;segment values
        mov  es,ax
        mov  sp,offset stkbase  ;use local stack
        mov  iobyte,0           ;clear iobyte
        push ds
        push es
        cld                     ;set interrupt 0 vector to
        mov  ax,0               ;  address trap routine
        mov  ds,ax
        mov  es,ax
        mov  int0_offst,offset int_trap
        mov  int0_segment,cs
        mov  di,4           ;propagate to remaining vectors
        mov  si,0
        mov  cx,510
    rep movs ax,ax
        mov  bdio,bdos_offset   ;correct bdos int vector
        pop  es
```

```
        pop  ds
        call con_init               ;initialize console
        xor  bx,bx                  ;get mass storage
    ini1:
        mov  ax,intbl[bx]           ;initlization table
        or   ax,ax                  ;quit if end of table
        jz   ini2
        push bx
        call ax                     ;call init entry
        pop  bx
        inc  bx                     ;step to next entry
        inc  bx
        jmp  ini1                   ;loop for next
    ini2:
        call login
        mov  bx,offset signon       ;print sign on msg
        call pmsg
        mov  cl,user                ;default to a: on coldstart
        jmp  ccp                    ;jump to cold entry of CCP


;------------------------------------------------------------
WBOOT:          ;enter CCP at command level

        jmp  ccp+6


;------------------------------------------------------------
CONST:          ;return console status

        in   al,cstat
        and  al,rdamsk
        jz   con1
        or   al,0ffh                ;return non-zero if rda
con1:   ret


;------------------------------------------------------------
CONIN:          ;get a character from console

        call CONST
        jz   CONIN              ;wait for RDA
        in   al,cdata
        and  al,7fh             ;read data & remove parity bit
        ret


;------------------------------------------------------------
CONOUT:         ;send a character to console

        in   al,cstat
        and  al,tbemsk              ;get console status
```

```
        jz    CONOUT
        mov   al,cl
        out   cdata,al              ;xmit buff is empty
        ret                         ;then return data

;----------------------------------------------------------------
LISTOUT:           ;send character to list device
                   ;not yet implemented

        ret

;----------------------------------------------------------------
PUNCH:             ;write character to punch device
                   ;not implemented

        ret

;----------------------------------------------------------------
READER:            ;get character from reader device
                   ;not implemented

        mov   al,1ah                ;return eof
        ret

;----------------------------------------------------------------
HOME:              ;move selected disk to trk 00
;                   one of seven device specific functions

        mov   track,0
        xor   bx,bx
        mov   bl,unit               ;get offset to actual device
        add   bx,bx
        call  hmtbl[bx]             ;call device code via tables
        ret
;
;
;----------------------------------------------------------------
SELDSK:            ;one of seven device specific functions
                   ;return pointer to appropriate 'disk
                   ;parameter block' (zero for bad unit no)
                   ;NOTE: nunits is defined in the .cfg file

        mov   unit,cl              ;save unit number
        mov   bx,0000h             ;ready for error return
        cmp   cl,nunits            ;return if beyond max unit
        jnb   sel1
        mov   bl,unit              ;get offset to actual device
        add   bx,bx
        call  dsktbl[bx]           ;call device code via tables
        xor   bx,bx
```

```
        mov  bl,unit            ;bx = cl * 16
        mov  cl,4
        shl  bx,cl
        mov  cx, offset dpbase ;bx += &dpbase
        add  bx,cx
    sell:
        ret

;----------------------------------------------------------------
SETTRK:             ;set track address
;                    one of seven device specific functions

        mov  track,cl
        xor  bx,bx
        mov  bl,unit            ;get offset to device
        add  bx,bx
        call trktbl[bx]         ;call device code via tables
        ret

;----------------------------------------------------------------
SETSEC:             ;set sector number
;                    one of seven device specific functions

        mov  sector,CL
        xor  bx,bx
        mov  bl,unit            ;get offset to device
        add  bx,bx
        call sectbl[bx]         ;call device code via tables
        ret

;----------------------------------------------------------------
SETDMA:             ;set DMA offset given by cx

        mov  dma_adr,cx
        ret


;----------------------------------------------------------------
READ:   ;read selected unit, track, sector to dma addr
        ;read and write operate by an indirect call
        ;through the appropriate table contained in
        ;the configuration file.  It is the programmers
        ;responsibility to ensure that the entry points
        ;in these tables match the unit type

        xor  bx,bx
        mov  bl,unit
        add  bx,bx
        call rdtbl[bx]         ;call device code via tables
        ret
```

```
;------------------------------------------------------------
WRITE:  ;write from dma address to selected
        ;unit, track, sector

        xor   bx,bx
        mov   bl,unit
        add   bx,bx
        call  wrtbl[bx]         ;call device code via tables
        ret


;------------------------------------------------------------
LISTST:          ;poll list device status
                 ;not implemented

        or    al,0ffh            ;return ready anyway or
        ret                      ;system may hang up


;------------------------------------------------------------
SECTRAN:         ;translate sector cx by table at [dx]
                 ;NOTE:  this routine is not adequate for
                 ;the case of >= 256 sectors per track
                 ;still it's better than DR's which is not
                 ;adequate for the no table case either

        mov   ch,0
        mov   bx,cx
        cmp   dx,0               ;check for no table case
        je    se1
        add   bx,dx              ;add sector to table addr
        mov   bl,[bx]            ;get logical sector
   se1:
        ret

;------------------------------------------------------------
SETDMAB:         ;set DMA segment given by cx

        mov   dma_seg,cx
        ret


;------------------------------------------------------------
GETSEGT:         ;return addr of physical memory table

        mov   bx,offset segtable
        ret


;------------------------------------------------------------
GETIOBF:         ;return iobyte value
                 ;note - this function and SETIOBF
```

```
                ;are OK but to implement the function
                ;the character IO entry point routines
                ;must be modified to redirect IO
                ;depending on the value of iobyte

        mov  al,iobyte
        ret


;-------------------------------------------------------------
SETIOBF:        ;set iobyte value

        mov  iobyte,cl
        ret


;*************************************************************
;                    SUBROUTINES
;*************************************************************



;-------------------------------------------------------------
int_trap:       ;interrupt trap - non interrupt
                ;driven system so should never get
                ;here - send mesage and halt

        cli                     ;block interrupts
        mov  ax,cs
        mov  ds,ax              ;get our data segment
        mov  bx,offset int_trp
        call pmsg
        hlt                     ;hardstop

;-------------------------------------------------------------
con_init:       ;initialize console driver
                ;actually done by the iSBC86/12a monitor

        ret


;-------------------------------------------------------------
pmsg:           ;send a message to the console

        mov  al,[bx]            ;get next char from message
        test al,al
        jz   pms1               ;if zero return
        mov  cl,al
        call CONOUT             ;print it
        inc  bx
        jmps pmsg               ;next character and loop
```

```
pmsl:
    ret

;*************************************************************
;           DISK SPECIFIC FUNCTION LABEL TABLES
;*************************************************************

;The included .cfg file below maps unit number to disk
;device type. It provides tables of entry point
;addresses for use by init, seldsk, seltrk selsec, home,
;read and write. These addresses must appear in the
;appropriate include file for the particular device type

            include cpmmast.cfg  ;read in label tables


;*************************************************************
;                 DISK INCLUDE FILES
;*************************************************************

;For each I/O device to be accessed by the operating
;system a separate file must be included. Within each file
;seven functions must be addressed and are the same ones
;mentioned in CPMMAST.CFG. The labels used to access these
;functions must be properly order in CPMMAST.CFG.

            include mb80dsk.a86     ;MBB-80 bubble memory
            include rxflop.a86      ;REMEX flopyy disks
            include rxhard.a86      ;REMEX hard disk

;*************************************************************
;                 RESOURCE ALLOCATION
;*************************************************************

;Low-level synchronization of access to the shared
;device. <sync.a86> must include the entry
;points defined in the cfg.files. These are
;called on initialization and before and after
;accessing the resource respectively.

            include sync.a86


;*************************************************************
;               DATA & LOCAL STACK AREA
;*************************************************************

        cseg $

signon  db      cr,lf,cr,lf
        db      cr,lf,lf,'
        if master
```

```
        db      'CPM/86 Master '
        endif
        if not master
        db      'CPM/86 Slave'
        endif
        db      cr,lf,lf,'                          Modified  '
        db      '   6 October 1982 by'
        db      cr,lf,lf,'                       Tom V. Almquist'
        db      ' and David S. Stevens',cr,lf,lf
        db      '           For use with a Bubble Memory and '
        db      'the REMEX Dataware House'
        db      cr,lf,0
int_trp db      cr,lf
        db      'Interrupt Trap Halt'
        db      cr,lf,0
iobyte  rb      1       ;character i/o redirection byte
unit    rb      1       ;selected unit
track   rb      1       ;selected track
sector  rb      1       ;selected sector
dma_adr rw      1       ;selected DMA address
dma_seg rw      1       ;selected DMA segment
loc_stk rw      32      ;local stack for initialization
stkbase equ     offset $

;system memory segment table

segtable        db 1            ;1 segment
                dw tpa_seg      ;1st seg starts after BIOS
                dw tpa_len      ;and extends to top of TPA
                dw 2000H
                dw 2000H

***********************************************************
                DISK DEFINITION TABLES
***********************************************************
;The included .lib file contains disk definition
;tables detailing disk characteristics for the bdos
;.lib files are generated by GENDEF from definition
;files and must comply with the allocations made in
;the corresponding configuration file. (Lable Tables)

        include cpmmast.lib     ;read in disk ief tables


;**********************************************************
;               END OF BIOS
;**********************************************************

lastoff equ     offset $
tpa_seg equ     (lastoff+0400h+15) / 16
tpa_len equ     1000h - tpa_seg
```

```
;*************************************************************
;                    PAGE ZERO TEMPLATE
;*************************************************************

                dseg    Ø       ;absolute low memory
                org     Ø       ;(interrupt vectors)
intØ_offst      rw      1
intØ_segment    rw      1
                rw      2*(bdos_int-1)
bdio            rw      1       ;bdos interrupt offset
bdis            rw      1       ;bdos interrupt segment


        end
```

APPENDIX C
PROGRAM LISTING OF CPMMAST.CFG

```
;Prog Name   : CPMMAST.CFG ( Master Configuration for CPM)
;Date        : 13 September 1982
;Written by  : Tom V. Almquist and David S. Stevens
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Kodres
;Purpose     : This code is an include file w/in CPMBIOS.A86.
;              It contains the device tables for access to
;              initialization, read, & write routines.


;------------------------------------------------------------
;                   DEFINE nunits

nunits  db 7     ;total number of mass storage  units

;------------------------------------------------------------
;                   INITIALIZATION TABLE
;
;intbl contains a sequence of addresses of initialization
;entry points to be called by the BIOS on entry after
;a cold boot.  The sequence is terminated by a zero entry

intbl   dw offset mb80dsk_init  ;initialize Bubble
        dw offset rxflop_init   ;initialize Remex
  if master
        dw offset initsync      ;initialize sync variables
        dw offset init_login    ;initialize login
  endif                         ;  procedures
        dw 0                    ;end of table


;------------------------------------------------------------
;                   READ TABLE

;rdtbl and wrtbl are sequences of length nunits, containing
;the addresses of the read and write entry point routines
;respectively which apply to the unit number corresponding
;to the position in the sequence.  These and the entry pts
;for initialization must correspond to those contained in
;the appropriate include files containing code specific
;to the devices.

rdtbl   dw offset mb80dsk_read  ;A: is a bubble memory
        dw offset rxflop_read   ;B: is Remex floppy disk 1
        dw offset rxflop_read   ;C: is Remex floppy disk 2
```

```
        dw offset rxhard_read   ;D: is Remex hard disk 0
        dw offset rxhard_read   ;E: is Remex hard disk 1
        dw offset rxhard_read   ;F: is Remex hard disk 2
        dw offset rxhard_read   ;G  is Remex hard disk 3


;----------------------------------------------------------
;                  WRITE TABLE

wrtbl   dw offset mb80dsk_write
        dw offset rxflop_write
        dw offset rxflop_write
        dw offset rxhard_write
        dw offset rxhard_write
        dw offset rxhard_write
        dw offset rxhard_write


;----------------------------------------------------------
;                      HOME TABLE
;
hmtbl   dw offset mb80dsk_home
        dw offset rxflop_home
        dw offset rxflop_home
        dw offset rxhard_home
        dw offset rxhard_home
        dw offset rxhard_home
        dw offset rxhard_home


;----------------------------------------------------------
;                      SELDSK TABLE

dsktbl  dw offset mb80dsk_seldsk
        dw offset rxflop_seldsk
        dw offset rxflop_seldsk
        dw offset rxhard_seldsk
        dw offset rxhard_seldsk
        dw offset rxhard_seldsk
        dw offset rxhard_seldsk


;----------------------------------------------------------
;                      SETTRK TABLE

trktbl  dw offset mb80dsk_settrk
        dw offset rxflop_settrk
        dw offset rxflop_settrk
        dw offset rxhard_settrk
        dw offset rxhard_settrk
        dw offset rxhard_settrk
        dw offset rxhard_settrk
```

```
;--------------------------------------------------------------
;                    SETSEC TABLE

sectbl  dw offset mb80dsk_setsec
        dw offset rxflop_setsec
        dw offset rxflop_setsec
        dw offset rxhard_setsec
        dw offset rxhard_setsec
        dw offset rxhard_setsec
        dw offset rxhard_setsec
```

APPENDIX D
PROGRAM LISTING OF MB80DSK.A86

```
;Prog Name    : MB80DSKA86 (BUBBLE MEMORY DISK)
;Date         : 24 Aug 1982
;Modified by  : Tom V. Almquist and David S. Stevens
;For          : Thesis (AEGIS Modeling Group)
;Advisor      : Professor Kodres
;Purpose      : This code is an include file w/in CPMBIOS.A86
;               It contains the code necessary to access the
;               bubble memory as a disk drive.

;+++++++++++++++++++++++ EQUATES +++++++++++++++++++++++++++

;------------------- Miscellaneous equates ---------------

mb_contbase     equ 8000H       ;controller base
addr_high_ram   equ 0f00H       ;high para user avail RAM
bdos_int_type   equ 224         ;reserved BDOS interrupt
sector_size     equ 128         ;CP/M logical dsk sector size

;-------- Magnetic bubble characteristics (MBB-80) --------

mb_buflen       equ 144   ;buffer length for MBB sector
mb_maxdevs      equ 7     ;bubble devices are #0-#7
mb_maxpages     equ 641   ;# of pages on each device
mb_maxsectors   equ 80    ;# of log. sectors on each dev
mb_pages_sec    equ 8     ;# of pages per logical sector
mb_pagesize     equ 18    ;bubble device page size
mb_skew         equ 12    ;skew factor for page xlation

;---- Magnetic bubble command bytes and masks (MBB-80) ----

mb_chkbusy_cmd  equ 020H  ;is controller busy ? status
mb_chkint_mask  equ 080H  ;mask to chk for MBB interupt
mb_inhint_cmd   equ 080H  ;interrupt inhibit/reset mask
mb_init_cmd     equ 01H   ;initialize the controller
mb_mpage_cmd    equ 010H  ;multi-page mode operation cmd
mb_read_cmd     equ 012H  ;multi-page read command
mb_reset_cmd    equ 040H  ;reset the controller
mb_write_cmd    equ 014H  ;multi-page write command
;
CSEG $
```

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                 DEVICE SPECIFIC ACCESS CODE
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;---------------------------------------------------------------
;initialize bubble                   ;called from INIT
                                     ;parm in - none
                                     ;parm out - none

mb80dsk_init:
        push     es
   init_mbb80:
        mov  ax,mb_contbase          ;controller base
        mov  es,ax                   ;address to es reg
        mov  ax,mb_maxpages          ;pgs per bubble dev
        mov  es:mbp_loopsize_lo,al
        mov  es:mbp_loopsize_hi,AE
        mov  es:mbp_pgsize_reg,mb_pagesize

        ;issue reset command to the controller

        mov  al,mb_reset_cmd         ;reset mask byte
        mov  es:mbp_cmnd_reg,al      ;issue reset cmd

        ;initialize each bubble device

        push cx                      ;save cx, outer counter
        mov  cx,mb_maxdevs+1         ;count for loop-# of devs
        mov  al,0                    ;device # to initialize
   For_each:
        mov  es:mbp_select_bub,al ;select each device
        mov  es:mbp_cmnd_reg,mb_init_cmd   ;init device
        push ax!push cx!push es      ;save bub#,counter,es
        call mbb80_wait              ;wait for controller
        pop  es! pop cx! pop ax      ;reset es,cnter,MBB#
        inc  al                      ;next device number
        loop for_each                ;dec cx, loop not zero
        pop  cx                      ;reset cx, outer cnter
        pop  es                      ;restore register
   Device_ret:
        ret

;---------------------------------------------------------------
;HOME BUBBLE                         ;called via home table

   mb80dsk_home:
        xor  cx,cx                   ;set track to zero
        call Settrk
        ret
```

```
;-----------------------------------------------------------
;SELECT BUBBLE DISK                ;called via seldsk table

   mb80dsk_seldsk:                 ;no special action required
        ret


;-----------------------------------------------------------
;SELECT BUBBLE TRACK               ;called via seltrk table

   mb80dsk_settrk:
        call mbb80_track_xlat
        ret

;-----------------------------------------------------------
;SET BUBLE SECTOR                  ;called via setsec table

   mb80dsk_setsec:                 ;no special action required
        ret



;-----------------------------------------------------------
;MBB80_READ      called via read table
                 ;reads a sector from bubble
                 ;parm in - none
                 ;parm out - status of the op in al.
                 ;  00= OK, FF= unsuccessful

mb80dsk_read:
        call request              ;get resource (SYNC.A86)
        push es                   ;save register
        call mbb80_sector_xlat    ;compute 1st page# of sect
        mov  ax,mb_contbase       ;addr of controller base
        mov  es,ax                ;load es to address bubble
        mov  es:mbp_cmnd_reg,mb_mpage_cmd ;multipage cmd
        mov  ax,mb_page_no        ;current page number
        mov  es:mbp_pagesel_lo,al ;page select lo byte
        mov  es:mbp_pagesel_hi,AH ;page select hi byte

        ;set number of pages to transfer = pages/sector

        mov  es:mbp_pagecnt_lo,mb_pages_sec  ;#pages xfer
        mov  es:mbp_pagecnt_hi,0 ;hi byte of # is 0

        ;set up dma address to receive data

        mov  cx,mb_buflen         ;count for loop-buffer size
        push ds                   ;save CP/M's ds
        mov  ax,dma_seg           ;get dma segment
        push ax                   ;save dma segment ds
```

```
        mov  bx,dma_adr           ;offset of dma area

        ;select bubble device and issue read command
        mov  al,mb_bub_no         ;current bubble number
        pop  ds                   ;local, readdr dma area
        mov  es:mbp_select_bub,al ;select current dev #
        mov  es:mbp_cmnd_reg,mb_read_cmd ;read from FIFO

   Read_int:
        mov  al,es:mbp_int_flag   ;get interrupt status
        and  al,mb_chkint_mask    ;interrupt set ?
        jz   Read_int             ;if zero, keep checking

        ;read enough from bubble sector to fill dma area?

        cmp  cx,(mb_buflen - sector_size) ;xfer enough?
        jnz  Read_one             ;if not, read another byte
        pop  ds                   ;restore CP/M's ds
        mov  bx,offset mb_overflow ;reset dest to ovrflow

        ;read from MBB FIFO buffer into dma area

   Read_one:
        mov  al,es:mbp_rdata_reg ;read a byte into accum
        mov  [bx],al              ;load accum into dma area
        inc  bx                   ;increment index
        loop Read_int             ;dec cx, loop if not zero
        push es                   ;save es for call
        call Mbb80_Wait           ;wait for controller
        pop  es                   ;restore es after call
        mov  es:mbp_cmnd_reg,mb_inhint_cmd ;clear int
        mov  al,0                 ;indicate no error
        push ax                   ;save status of read
        call release              ;free resource (SYNC.A86)
        pop  ax                   ;restore registers
        pop  es
        ret


;----------------------------------------------------------
;MBB80_WRITE             called via write table
                         ;writes a sector to bubble
                         ;parm in - none
                         ;parm out - status of the op in al
                         ;00 = OK, FF = unsuccessful

mb80dsk_write:
        mov  al,0                ;bubble logical drive
        cmp  al,user             ;is user logged in on mb80
        jnz  mbwrt_err
        call request             ;get resource (SYNC.A86)
        push es                  ;save register
```

```
        call Mbb80_Sector_Xlat  ;get 1st page# of sector
        mov  ax,mb_contbase     ;address of controller base
        mov  es,ax              ;load es to address bubble
        mov  es:mbp_cmnd_reg,mb_mpage_cmd;multpg mode cmd
        mov  ax,mb_page_no           ;current page number
        mov  es:mbp_pagesel_lo,al    ;page select lo byte
        mov  es:mbp_pagesel_hi,AH    ;page select hi byte

        ;set number of pages to transfer = pages/sector

        mov  es:mbp_pagecnt_lo,mb_pages_sec ;#pages to xfer
        mov  es:mbp_pagecnt_hi,0      ;hi byte of # is zero

        ;set up dma address for transfer

        mov  cx,mb_buflen-1     ;count for loop-write
        push ds                 ;save CP/M's ds
        mov  ax,dma_seg         ;get dma segment
        push ax                 ;save dma segment ds
        mov  bx,dma_adr         ;address of dma area

        ;select bubble device and issue write cmd

        mov  al,mb_bub_no           ;current bubble number
        mov  es:mbp_select_bub,al  ;select current dev #
        pop  ds                    ;readdr dma area
        mov  al,[bx]               ;load first byte
        mov  es:mbp_wdata_reg,al   ;write byte to MBB buff
        inc  bx                    ;increment index
        mov  es:mbp_cmnd_reg,mb_write_cmd;send write to MBB

        ;wait for interrupt from controller

Write_int:
        mov  al,es:mbp_int_flag    ;get interrupt status
        and  al,mb_chkint_mask     ;interrupt set ?
        jz   Write_int             ;if zero, keep checking

        ;write into MBB FIFO buffer from dma area

        mov  al,[bx]          ;byte from dma to al
        mov  es:mbp_wdata_reg,al ;write byte to MBB buff
        inc  bx               ;increment index
        loop Write_int        ;dec cx, loop if not zero
        pop  ds               ;restore CP/M's ds
        push es               ;save es for call
        call Mbb80_Wait       ;wait for controller
        pop  es               ;restore es after call
        mov  es:mbp_cmnd_reg,mb_inhint_cmd;clear contint
        mov  al,0             ;return success code
        push ax               ;save success code
        call release          ;free resource (SYNC.A86)
```

```
        pop  ax
        pop  es                 ;restore register
        jmp  mbwrt_ret
    mbwrt_err:
        mov  bx,offset mbwrt_msg
        call pmsg
        mov  al,0ffh            ;error returned to  CP/M
    mbwrt_ret:
        ret


;++-++++++++++++++++-++++++++++++++-++++++++++++++++++++++
;                  BUBBLE SUBROUTINES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;---------------------------------------------------------
;MBB80_SECTOR_XLAT     called from: Mbb80_Read, Mbb80_Write.
                       ;computes 1st page# for a given sector
                       ;on a single chip. Based on 80 sectors
                       ;on each chip - sector = 128 bytes.
                       ;parm in - none, works on sector
                       ;parm out - none, updates mb_page_no

Mbb80_Sector_Xlat:
        xor  ax,ax              ;set ax to 0 to hold page#
        xor  cx,cx              ;clear cx for counter
        mov  CL,sector          ;ctr for translation loop
        xor  DX,DX              ;clear DX
        mov  DL,mb_sector       ;sect# for 1st sect on trk
        add  cx,DX              ;add 1st sect# to log sect#
        dec  CL                 ;subtract 1 for the loop
        jz   Mbb80_sx_exit      ;sect 1 is page 0, no xlat
   Add_skew:
        add  ax,mb_skew         ;add skew between pages
        clc                     ;clear carry
        sbb  ax,mb_maxpages     ;mod to # of pages
        jae  Dec_sector         ;jump if positive (CF=0)
        add  ax,mb_maxpages     ;went (-), add back #pages
   Dec_sector:
        loop Add_skew           ;dec sector#,add skew again
   Mbb80_sx_exit:
        mov  mb_page_no,ax      ;store page number
        ret
;
;
;---------------------------------------------------------
;MBB80_TRACK_XLAT       called from: SETTRK.
                        ;computes bubble # from track #. Gets
                        ;first bubble sector (1-80) for that
                        ;track for later conversion to page #.
                        ;parm in - none, works on track.
                        ;prm out - loads mb_bub_no,mb_sector
```

```
Mbb80_Track_Xlat:
        xor  bx,bx                   ;clear bx for add
        mov  BL,track                ;load track - index
        add  BL,BL                   ;double track# for index
        mov  ax,mb_track_table[bx] ;get word from table
        mov  mb_bub_no,AH           ;low byte = bubb device#
        mov  mb_sector,al           ;high byte = 1st sector#
        ret


;-------------------------------------------------------------
;MBB80_WAIT         called from: Mbb80_Init, Mbb80_Read,
                             ;Mbb80_Write.
                    ;checks status of MBB cont for busy
                    ;keeps checking (wait) until not busy
                    ;parm in - none
                    ;parm out - none

Mbb80_Wait:
        mov  ax,mb_contbase         ;address of cont base
        mov  es,ax                  ;load es to addr bubble
   See_zero:
        mov  al,es:mbp_status_reg ;get status register
        and  al,mb_chkbusy_cmd     ;is it all zeros ?
        jz   See_zero              ;if so, keep checking
   Cont_busy:
        mov  al,es:mbp_status_reg ;get status register
        and  al,mb_chkbusy_cmd     ;see if busy, and to mask
        jnz  Cont_busy             ;if busy, check again
        ret



;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                     DATA SEGMENT AREA
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


;-----------------Bubble Variables-------------------------
mbwrt_msg        db  cr,lf,'Write Access Not Permited'
                 db  ' On This Drive.',0
mb_bub_no        rb  1      ;bubble device number 0-7
mb_overflow      rb  (Mb_buflen - sector_size) ;read overflw
mb_page_no       rw  1      ;bubble page number
mb_sector        rb  1      ;bubble sector number (1-80)

;Each entry in the track table corresponds to one of the
;24 tracks on the MBB-80. The 1st byte in each entry is the
;bubble number; the 2nd byte in each entry is the starting
;sector number for that track on that bubble device.
mb_track_table  dw  0000H,201aH,0034H,0100H,011aH,0134H
```

```
                dw   0200H,021aH,0234H,0300H,031aH,0334H
                dw   0400H,041aH,0434H,0500H,051aH,0534H
                dw   0600H,061aH,0634H,0700H,071aH,0734H
;
esEG
;
mbp_pagesel_lo  rb  1      ;ls byte for page select, (0)
mbp_pagesel_hi  rb  1      ;ms 2 bits for page select, (1)
mbp_cmnd_reg    rb  1      ;command register, (2)
mbp_rdata_reg   rb  1      ;read data register, (3)
mbp_wdata_reg   rb  1      ;write data register, (4)
mbp_status_reg  rb  1      ;status register, (5)
mbp_pagecnt_lo  rb  1      ;ls byte for page counter, (6)
mbp_pagecnt_hi  rb  1      ;ms 2 bits for page counter, (7)
mbp_loopsize_lo rb  1      ;ls byte for minor loop size,(8)
mbp_loopsize_hi rb  1      ;ms 2 bits for min loop size,(9)
                rw  1      ;internal use(page pos), (A,B)
mbp_pgsize_reg  rb  1      ;page size register, (C)
                rw  1      ;TI use only, (D,E)
mbp_select_bub  rb  1      ;two uses: select bubble dev (F)

mbp_int_flag    equu     mbp_select_bub ;interrupt flag (F)
```

APPENDIX E
PROGRAM LISTING OF RXFLOP.A86

```
;Prog Name   : RXFLOP.A86 (REMEX FLOPPY DISK
;                                ACCESS CODE)
;Date        : 9 October 1982
;Written by  : Tom V. Almquist and David S. Stevens
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Kodres
;Purpose     : This code is an include file w/in CPMBIOS.A86.
;              It contains the code necessary to access the
;              Remex floppy disk drives. I/O done through
;              common memory. This configuration is set for
;              CP/M logical drives 1 (B:) and 2 (C:). To
;              alter, change code in READ and WRITE routines.


;+++++++++++++++++++ Equates ++++++++++++++++++++++++++++++

;--- Disk Controller command bytes and masks (REMEX)    ---

   dk_rdy_mask       equ 08H
   dk_rd_cmd1        equ 1011H      ;read command
   dk_rd_cmd2        equ 1012H
   dk_wr_cmd1        equ 1021H      ;write command
   dk_wr_cmd2        equ 1022H
   tries             equ 10
   drive2            equ 2          ;CPM logical dsk # for
                                    ;drive 2


;-------- REMEX Interface  Controller Ports ------------

   cmd_reg           equ 70H        ;ctrler's base in CP/M-86
   status_reg        equ 71H
   p_addr_lo         equ 72H
   p_addr_hi         equ 73H


;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                 CPM DEVICE SPECIFIC CODE
;        entered via label tables in CPMMAST.CFG
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        cseg $

;----------------------------------------------------------
rxflop_init:
```

```
        ret             ;no special action required


;-----------------------------------------------------------------
rxflop_home:
        ret             ;no special action required


;-----------------------------------------------------------------
rxflop_seldsk:
        ret             ;no special action required


;-----------------------------------------------------------------
rxflop_settrk:
        ret             ;no special action required


;-----------------------------------------------------------------
rxflop_setsec:
        ret             ;no special action required


;-----------------------------------------------------------------
rxflop_read:

        mov     rwdir,0
        call    request         ;get resource (SYNC.A86)
        cmp     unit,drive2     ;CP/M logical disk No. for
        jz      rd1             ;Remex floppy drive 2 (C:)
        mov     bx,dk_rd_cmd1   ;set up to read drive 1 (B:)
        jmps    rd2
    rd1:
        mov     bx,dk_rd_cmd2   ;set up to read drive 2
    rd2:
        call    build_packet
        call    send_packet     ;perform the read
        call    xfr_buffer      ;xfr CPM buffer into memory
        call    release         ;free resource (SYNC.A86)
        mov     al,result       ;return success/failure code
        ret


;-----------------------------------------------------------------
rxflop_write:

        mov     rwdir,1
        call    request         ;request ticket number
        cmp     unit,drive2     ;CP/M logical disk No. for
        jz      wrt1            ;Remex floppy drive 2 (C:)
```

```
        mov     bx,dk_wr_cmd1   ;setup write to drive 1 (B:)
        jmps    wrt2
   wrt1:
        mov     bx,dk_wr_cmd2   ;set up to write drive 2
   wrt2:
        call    build_packet
        call    xfr_buffer
        call    send_packet
        call    release         ;free resource (SYNC.A86)
        mov     al,result       ;return success/failure code
        ret


;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                REMEX FLOPPY DISK SUBROUTINES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

build_packet:
        push    es              ;save es register
        mov     ax,cmemseg      ;set up es to address common
        mov     es,ax           ;memory  E000:
        mov     p_modifiers,bx  ;enter read code in packet
        mov     p_status,0      ;clear packet status word
        mov     ax,0000H        ;clear register
        mov     al,track        ;get track #
        mov     p_track_no,ax   ;enter track # in packet
        mov     ax,0000H        ;set head no. to 0
        add     al,sector       ;set sector no.
        mov     p_head_sect,ax  ;put head & sec # in packet
        mov     p_mem_addr,0100h ;address of CPM buffer
        mov     p_msb,000eh     ;CPM buffer msb
        mov     p_word_count,64 ;# of 16 bit words
        pop     es
        ret

;-----------------------------------------------------------
send_packet:

        push    es
        mov     ax,cmemseg   ;common memory segement = E000
        mov     es,ax
        mov     dk_cnt,tries  ;load count for retries
   send1:
        in      al,status_reg
        and     al, dk_rdy_mask ;check interface ready
        cmp     al,08H          ;is it ready?
        jne     send1           ;if not ready repeat
        mov     al,1cH
        out     cmd_reg,al      ;load extended address
        mov     ax,0024h        ;packet offset
        out     p_addr_lo,al    ;transfer low byte out
        mov     al,ah
```

```
        out     p_addr_hi,al      ;transfer hi byte out
check_result:
        mov     ax,p_status       ;load status word
        cmp     ax,0001H          ;check for success
        je      success_read
        cmp     ax,0000H          ;check for failure
        jne     retry
        jmps    check_result
    retry:
        mov     dk_err_code,al    ;save error code
        mov     ax,0              ;clear status word
        dec     dk_cnt            ;reduce retry count
        jnz     send_packet       ;if <> 0 try again
        mov     result,0FFH               ;return failure code
        jmps    dk_execute_ret
    success_read:
        mov     result,00H                ;return success code
    dk_execute_ret:
        pop     es
        ret

;-----------------------------------------------------------
xfr_buffer:                      ;get data from common memory
                                 ;and load into local memory

        push es ! push ds
        mov  es,dma_seg
        mov  di,dma_adr
        mov  ax,cmemseg
        mov  ds,ax
        mov  si,0100h
        mov  cx,64
        cmp  rwdir,0
        jz   xfr
        xchg si,di                ;set up for write operation
        mov  ax,ds
        mov  es,ax
        mov  ds,dma_seg
xfr:
        cld
        rep  movs ax,ax           ;move as 16-bit words
        pop  ds ! pop es
        ret

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                    Data Area
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;---------------- Remex Interface Packet----------------
;packet located in common memory at E000:0004

                eseg
```

```
                 org 0004h  ;offset of packet

p_modifiers      rw  1      ;function & logical unit
p_status         rw  1      ;returned status
p_track_no       rw  1      ;selected track number
p_head_sect      rw  1      ;selected head/sector number
p_mem_addr       rw  1      ;buffer address
p_msb            rw  1      ;extended bits of buffer address
p_word_count     rw  1      ;size of data block

;--------------------Misc Variables---------------------

                 cseg $

dk_err_code      db  00H    ;returned Remex error code
dk_cnt           db  00H
result           rb  1
rwdir            rb  1      ;0 = read ; 1 = write
```

APPENDIX F
PROGRAM LISTING OF RXHARD.A86

```
;Prog Name   : RXHARD2.A86 (REMEX HARD DISK ACCESS CODE)
;Date        : 13 October 1982
;Modified    : Transfer Thru Common Memory/Ticket Sync
;Written by  : Tom V. Almquist and David S. Stevens
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Kodres
;Purpose     : This code is an include file w/in CPMBIOS.A86.
;              It contains the code necessary to access the
;              REMEX hard disk drive.
;
;
;----------------- Equates ---------------------------

;--- Disk Controller command bytes and masks (REMEX)   ---

hdk_rdy_mask     equu 08H
hdk_rd_cmd       equ 1010H      ;read command
hdk_wr_cmd       equ 1020H      ;write command
hdk_tries        equ 10
head0            equ 3          ;CP/M logical dsk# for head
                                ;0 of REMEX hard disk
pstrf            equ 9          ;print string function

;--------- REMEX Interface  Controller Ports -------------

hdk_CMD_reg      equ 70H        ;ctrler's base in CP/M-86
hdk_status_reg   equ 71H
hdk_addr_lo      equ 72H
hdk_addr_hi      equ 73H

;------------------Blocking/Deblocking-----------------

una              equ byte ptr [BX] ;name for byte at BX
blksiz           equ 16384         ;CP/M allocation size
hstsiz           equ 512           ;host disk sector size
hstspt           equ 39            ;host disk sectors/trk
hstblk           equ hstsiz/128    ;CP/M sects/host buff
secshf           equ 2             ;log2(hstblk)
cpmspt           equ hstblk * hstspt ;CP/M sectors/track
secmsk           equ hstblk-1      ;sector mask
wrall            equ 0             ;write to allocated
wrdir            equ 1             ;write to directory
wrual            equ 2             ;write to unallocated
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                    DEVICE SPECIFIC CODE
;      entered from the main CPMBIOS via label tables
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

CSEG $
;--------------------------------------------------------------
; INIT                          ;called from INIT
rxhard_init:
        ret


;--------------------------------------------------------------
;HOME                           entered via home label table
Rxhard_home:
        mov     al,hstwrt       ;check for pending write
        test    al,al
        jnz     homed
        mov     hstact,0        ;clear host active flag
    homed:
        ret


;--------------------------------------------------------------
;SELECT DISK                    entered via seldsk label table
Rxhard_seldsk:
        mov     cl,unit
        mov     sekdsk,cl
        test    dl,1            ;1st activation of disk?
        jnz     cont1           ;no
        mov     hstact,0        ;yes
        mov     unacnt,0
    cont1:
        ret


;--------------------------------------------------------------
;SELECT TRACK                   enterd via seltrk label table
Rxhard_settrk:
        mov     sektrk,cx
        ret


;--------------------------------------------------------------
;SELECT SECTOR                   entered via selsec label table

Rxhard_setsec:
        mov     seksec,cl
        ret
```

```
;------------------------------------------------------------------
;READ                        entered via read label table

Rxhard_read:                          ;read selected CP/M sector
        mov     unacnt,0              ;clear unallocated counter
        mov     readop,1              ;read operation
        mov     rsflag,1              ;must read data
        mov     wrtype,wrual          ;treat as unalloc
        jmp     rwoper                ;to perform the read

;------------------------------------------------------------------
;WRITE                       enter via write label table

Rxhard_write:                         ;write selected CP/M sector
        mov     readop,0              ;write operation
        mov     wrtype,cl
        cmp     cl,wrual              ;write unallocated?
        jnz     chkuna                ;check for unalloc

                ;write to unallocated, set parameters
        mov     unacnt,(blksiz/128) ;next unalloc recs
        mov     al,sekdsk             ;disk to seek
        mov     unadsk,al             ;unadsk = sekdsk
        mov     ax,sektrk
        mov     unatrk,ax             ;unatrk = sektrk
        mov     al,seksec
        mov     unasec,al             ;unasec = seksec

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;               BLOCKING & DEBLOCKING SUBROUTINES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Chkuna:         ;check for write to unallocated sector
        mov     bx,offset unacnt;point "UNA" at UNACNT
        mov     al,una
        test    al,al                 ;any unalloc remain?
        jz      alloc                 ;skip if not

;more unallocated records remain
        dec     al                    ;unacnt = unacnt-1
        mov     una,al
        mov     al,sekdsk             ;same disk?
        mov     bx,offset unadsk
        cmp     al,una                ;sekdsk = unadsk?
        jnz     alloc                 ;skip if not

                ;disks are the same
        mov     AX, unatrk
        cmp     AX, sektrk
        jnz     alloc                 ;skip if not

                ;tracks are the same
```

```
        mov     al,seksec           ;same sector?
        mov     bx,offset unasec ;point una at unasec
        cmp     al,una              ;seksec = unasec?
        jnz     alloc               ;skip if not

;match, move to next sector for future ref
        inc     una                 ;unasec = unasec+1
        mov     al,una              ;end of track?
        cmp     al,cpmspt           ;count CP/M sectors
        jb      noovf               ;skip if below

;overflow to next track
        mov     una,0               ;unasec = 0
        inc     unatrk              ;unatrk=unatrk+1

    noovf:      ;match found, mark as unnecessary read
        mov     rsflag,0            ;rsflag = 0
        jmps    rwoper              ;to perform the write

    alloc:    ;not an unallocated record, requires pre-read
        mov     unacnt,0            ;unacnt = 0
        mov     rsflag,1            ;rsflag = 1
                                    ;drop through to rwoper

;Common code for READ and WRITE follows

rwoper: ;enter here to perform the read/write
        mov     erflag,0            ;no errors (yet)
        mov     al, seksec          ;compute host sector
        sub     al,1
        mov     cl, secshf
        shr     al,cl
        mov     sekhst,al           ;host sector to seek

;active host sector?
        mov     al,1
        xchg    al,hstact           ;always becomes 1
        test    al,al               ;was it already?
        jz      filhst              ;fill host if not

;host buffer active, same as seek buffer?
        mov     al,sekdsk
        cmp     al,hstdsk           ;sekdsk = hstdsk?
        jnz     nomatch

                ;same disk, same track?
        mov     ax,hsttrk
        cmp     ax,sektrk           ;host trk same as seek trk
        jnz     nomatch

;same disk, same track, same buffer?
        mov     al,sekhst
```

```
        cmp     al,hstsec          ;sekhst = hstsec?
        jz      match              ;skip if match

    nomatch:    ;proper disk, but not correct sector
        mov     al,     hstwrt
        test    al,al              ;"dirty" buffer ?
        jz      filhst             ;no, don't need to write
        call    writehst           ;yes, clear host buff

    filhst:     ;may have to fill the host buffer
        mov     al,sekdsk          ! mov hstdsk,al
        mov     ax,sektrk          ! mov hsttrk,ax
        mov     al,sekhst          ! mov hstsec,al
        mov     al,rsflag
        test    al,al              ;need to read?
        jz      filhst1
        call    readhst

    filhst1:
        mov     hstwrt,Ø           ;no pending write

    match:
        ;copy data to or from buffer depending on "readop"
        mov     al,seksec          ;mask buffer number
        sub     al,1
        and     ax,secmsk          ;least signif bits masked
        mov     cl,7               ;shift lsft 7
        shl     ax,cl              ;(* 128 = 2**7)

;ax has relative host buffer offset

        add     ax,offset hstbuf ;ax has buffer address
        mov     si,ax              ;put in source index reg
        mov     di,dma_adr         ;user buff is dest if readop
        push    DS
        push    ES                 ;save segment registers
        mov     ES,dma_seg         ;set destseg to the user seg
                                   ;SI/DI and DS/ES is swapped
                                   ;if write op
        mov     cx,128/2           ;length of move in words
        mov     al,readop
        test    al,al              ;which way?
        jnz     rwmove             ;skip if read

                ;write operation, mark and switch direction
        mov     hstwrt,1           ;hstwrt = 1 (dirty buffer )
        xchg    si,di              ;source/dest index swap
        mov     ax,DS
        mov     ES,ax
        mov     DS,dma_seg         ;setup DS,ES for write

    rwmove:
```

```
        cld
    rep movs    AX,AX              ;move as 16 bit words
        pop     ES
        pop     DS                 ;restore segment registers

                ;data has been moved to/from host buffer
        cmp     wrtype,wrdir       ;write type to directory?
        mov     al,erflag          ;in case of errors
        jnz     return_rw          ;no further processing

                ;clear host buffer for directory write
        test    al,al              ;errors?
        jnz     return_rw          ;skip if so
        mov     hstwrt,0           ;buffer written
        call    writehst
        mov     al,erflag
    return_rw:
        ret


;-----------------------------------------------------------
read_hst:

        mov     hdk_rwdir,0
        call    request             ;get resource (SYNC.A86)
        mov     bx,hdk_rd_cmd
        call    hdk_build_packet
        call    hdk_send_packet     ;perform the read
        call    hdk_xfr_buffer
        call    release             ;free resource (SYNC.A86)
        mov     al,hdk_result       ;ret success/failure code
        ret


;------------------------------------------------------------
write_hst:

        mov     hdk_rwdir,1
        mov     al,hst_dsk
        cmp     al,user
        jnz     wrt_err
        call    request             ;get resource (SYNC.A86)
        mov     bx,hdk_wr_cmd       ;set up write to hard disk
        call    hdk_build_packet
        call    hdk_xfr_buffer
        call    hdk_send_packet
        call    release             ;free resource (SYNC.A86)
        mov     al,hdk_result       ;ret success/failure code
        jmp     wrt_ret
   wrt_err:
        mov     bx,offset wrtmsg
        call    pmsg
```

```
        mov     al,0ffh         ;return error to CP/M
   wrt_ret:
        ret


;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;               REMEX HARD DISK SUBROUTINES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

hdk_build_packet:   ;packet built in common memory

        push    es
        mov     ax,cmemseg
        mov     es,ax
        mov     hdk_modifiers,bx ;enter read code in packet
        mov     hdk_status,0000H ;clear packet status word
        mov     AX,0000H        ;clear register
        mov     ax,hst_trk      ;get track no.
        mov     hdk_track_no,AX ;enter track no. in packet
        mov     AX,0000H        ;clear register
        mov     ah,hst_dsk
        sub     ah,head0        ;determine head #
        mov     AL,hst_sec      ;set sector #
        add     ax,1
        mov     hdk_head_sect,AX ;load in packet
        mov     hdk_mem_addr,0100h  ;address of CP/M buffer
        mov     hdk_msb,000eh       ;common memory seg
        mov     hdk_word_cnt,256 ;# of 16 bit words
        pop     es
        ret

;-----------------------------------------------------------------
hdk_send_packet:

        push    es
        mov     ax,cmemseg
        mov     es,ax
        mov     hdk_cnt,hdk_tries ;load count for retries
   send_hdk_packet:
        in      AL,hdk_status_reg
        and     AL,hdk_rdy_mask ;check interface ready
        cmp     AL,08H          ;is it ready?
        jne     send_hdk_packet ;if not ready repeat
        mov     al,1ch
        out     hdk_cmd_reg,AL  ;load extended address
        mov     ax,0004h
        out     hdk_addr_lo,AL  ;transfer low byte out
        mov     AL,AH
        out     hdk_addr_hi,AL  ;transfer hi byte out
    check_hdk_result:
        mov     ax,hdk_status   ;load status word
        cmp     AX,0001H        ;check for success
```

```
        je      hdk_success_read
        cmp     AX,0000H        ;check for failure
        jne     hdk_retry
        jmps    check_hdk_result
    hdk_retry:
        mov     hdk_err_code,AL ;save error code
        mov     hdk_status,0            ;clear status word
        dec     hdk_cnt         ;reduce retry count
        jnz     send_hdk_packet ;if <> 0 try again
        mov     hdk_result,0FFH         ;return failure code
        jmps    hdk_execute_ret
   hdk_success_read:
        mov     hdk_result,00H          ;return success code
   hdk_execute_ret:
        pop     es
        ret

;------------------------------------------------------------
hdk_xfr_buffer:                         ;transfer data from common
                                        ;memory to local memory

        push    es ! push ds
        mov     ax,cs
        mov     es,ax
        mov     di,offset hstbuf
        mov     ax,cmemseg
        mov     ds,ax
        mov     si,0100h
        mov     cx,256
        cmp     hdk_rwdir,0
        jz      hdk_xfr
        xchg    si,di
        mov     ax,ds
        mov     es,ax
        mov     ax,cs
        mov     ds,ax
   hdk_xfr:
        cld
        rep     movs ax,ax
        pop     ds ! pop  es
        ret

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                       Data Segment Area
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;----------------- Remex Interface Packet-----------------
;packet built in common memory at E000:0004

                eseg
                org 0004h  ;offset of packet
```

```
hdk_modifiers   rw  1       ;function & logical unit
hdk_status      rw  1       ;returned status
hdk_track_no    rw  1       ;selected track number
hdk_head_sect   rw  1       ;selected head/sector number
hdk_mem_addr    rw  1       ;buffer address
hdk_msb         rw  1       ;extended bits of buffer address
hdk_word_cnt    rw  1       ;size of data block

                cseg $

;-------------------Misc Variables---------------------

hdk_err_code    db  00H     ;returned Remex error code
hdk_cnt         db  00H
hdk_result      rb  1       ;success/failure code
hdk_rwdir       rb  1

sek_dsk rb      1           ;seek disk number
sek_trk rw      1           ;seek track number
sek_sec rb      1           ;seek sector number
hst_dsk rb      1           ;host disk number
hst_trk rw      1           ;host track number
hst_sec rb      1           ;host sector number
sek_hst rb      1           ;seek shr secshf
hst_act rb      1           ;host active flag
hst_wrt rb      1           ;host written flag
una_cnt rb      1           ;unalloc rec cnt
una_dsk rb      1           ;last unalloc disk
una_trk rw      1           ;last unalloc track
una_sec rb      1           ;last unalloc sector
erflag  rb      1           ;error reporting
rsflag  rb      1           ;read sector flag
readop  rb      1           ;1 if read operation
wrtype  rb      1           ;write operation type
dma_off rw      1           ;last dma offset
hstbuf  rb      hstsiz      ;host buffer

wrtmsg  db      cr,lf,'Write Access Not Permitted On This'
        db      ' Drive',0
```

APPENDIX G
PROGRAM LISTING OF CPMMAST.DEF

The following disk definition statements were used in this thesis. The command "GENDEF CPMMAST.DEF" is executed to produce CPMMAST.LIB which must be assembled into the BIOS using an "include" command.

```
disks 7
diskdef 0,1,26,0,1024,71,32,0,2
diskdef 1,1,26,6,1024,243,64,64,2
diskdef 2,1
diskdef 3,1,156,0,16384,275,128,0,1
diskdef 4,3
diskdef 5,3
diskdef 6,3
endef
```

APPENDIX H
PROGRAM LISTING OF CPMMAST.LIB

When GENDEF is executed using CPMMAST.DEF as the source file, CPMMAST.LIB is created. The listing which follow is the code generated by GENDEF and must be assembled into the BIOS with an "include" command.

```
;
;               DISKS 7
dpbase  equ     $               ;Base of Disk Parameter Blocks
dpe0    dw      xlt0,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb0     ;Dir Buff, Parm Block
        dw      csv0,alv0       ;Check, Alloc Vectors
dpe1    dw      xlt1,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb1     ;Dir Buff, Parm Block
        dw      csv1,alv1       ;Check, Alloc Vectors
dpe2    dw      xlt2,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb2     ;Dir Buff, Parm Block
        dw      csv2,alv2       ;Check, Alloc Vectors
dpe3    dw      xlt3,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb3     ;Dir Buff, Parm Block
        dw      csv3,alv3       ;Check, Alloc Vectors
dpe4    dw      xlt4,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb4     ;Dir Buff, Parm Block
        dw      csv4,alv4       ;Check, Alloc Vectors
dpe5    dw      xlt5,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb5     ;Dir Buff, Parm Block
        dw      csv5,alv5       ;Check, Alloc Vectors
dpe6    dw      xlt6,0000h      ;Translate Table
        dw      0000h,0000h     ;Scratch Area
        dw      dirbuf,dpb6     ;Dir Buff, Parm Block
        dw      csv6,alv6       ;Check, Alloc Vectors
;               DISKDEF 0,1,26,0,1024,71,32,0,2
dpb0    equ     offset $        ;Disk Parameter Block
        dw      26              ;Sectors Per Track
        db      3               ;Block Shift
        db      7               ;Block Mask
```

```
        db      0                   ;Extnt Mask
        dw      70                  ;Disk Size - 1
        dw      31                  ;Directory Max
        db      128                 ;Alloc0
        db      0                   ;Alloc1
        dw      0                   ;Check Size
        dw      2                   ;Offset
xlt0    equ     offset $            ;Translate Table
        db      1,2,3,4
        db      5,6,7,8
        db      9,10,11,12
        db      13,14,15,16
        db      17,18,19,20
        db      21,22,23,24
        db      25,26
als0    equ     9                   ;Allocation Vector Size
css0    equ     0                   ;Check Vector Size
;               DISKDEF 1,1,26,6,1024,243,64,64,2
dpb1    equ     offset $            ;Disk Parameter Block
        dw      26                  ;Sectors Per Track
        db      3                   ;Block Shift
        db      7                   ;Block Mask
        db      0                   ;Extnt Mask
        dw      242                 ;Disk Size - 1
        dw      63                  ;Directory Max
        db      192                 ;Alloc0
        db      0                   ;Alloc1
        dw      16                  ;Check Size
        dw      2                   ;Offset
xlt1    equ     offset $            ;Translate Table
        db      1,7,13,19
        db      25,5,11,17
        db      23,3,9,15
        db      21,2,8,14
        db      20,26,6,12
        db      18,24,4,10
        db      16,22
als1    equ     31                  ;Allocation Vector Size
css1    equ     16                  ;Check Vector Size
;               DISKDEF 2,1
dpb2    equ     dpb1                ;Equivalent Parameters
als2    equ     als1                ;Same Allocation Vector Size
css2    equ     css1                ;Same Checksum Vector Size
xlt2    equ     xlt1                ;Same Translate Table
;               DISKDEF 3,1,156,0,16384,275,128,0,1
dpb3    equ     offset $            ;Disk Parameter Block
        dw      156                 ;Sectors Per Track
        db      7                   ;Block Shift
        db      127                 ;Block Mask
        db      7                   ;Extnt Mask
        dw      274                 ;Disk Size - 1
        dw      127                 ;Directory Max
```

```
         db      128                 ;Alloc0
         db      0                   ;Alloc1
         dw      0                   ;Check Size
         dw      1                   ;Offset
xlt3     equ     offset $            ;Translate Table
         db      1,2,3,4
         db      5,6,7,8
         db      9,10,11,12
         db      13,14,15,16
         db      17,18,19,20
         db      21,22,23,24
         db      25,26,27,28
         db      29,30,31,32
         db      33,34,35,36
         db      37,38,39,40
         db      41,42,43,44
         db      45,46,47,48
         db      49,50,51,52
         db      53,54,55,56
         db      57,58,59,60
         db      61,62,63,64
         db      65,66,67,68
         db      69,70,71,72
         db      73,74,75,76
         db      77,78,79,80
         db      81,82,83,84
         db      85,86,87,88
         db      89,90,91,92
         db      93,94,95,96
         db      97,98,99,100
         db      101,102,103,104
         db      105,106,107,108
         db      109,110,111,112
         db      113,114,115,116
         db      117,118,119,120
         db      121,122,123,124
         db      125,126,127,128
         db      129,130,131,132
         db      133,134,135,136
         db      137,138,139,140
         db      141,142,143,144
         db      145,146,147,148
         db      149,150,151,152
         db      153,154,155,156
als3     equ     35                  ;Allocation Vector Size
css3     equ     0                   ;Check Vector Size
;                DISKDEF 4,3
dpb4     equ     dpb3                ;Equivalent Parameters
als4     equ     als3                ;Same Allocation Vector Size
css4     equ     css3                ;Same Checksum Vector Size
xlt4     equ     xlt3                ;Same Translate Table
;                DISKDEF 5,3
```

```
dpb5    equ     dpb3            ;Equivalent Parameters
als5    equ     als3            ;Same Allocation Vector Size
css5    equ     css3            ;Same Checksum Vector Size
xlt5    equ     xlt3            ;Same Translate Table
;               DISKDEF 6,3
dpb6    equ     dpb3            ;Equivalent Parameters
als6    equ     als3            ;Same Allocation Vector Size
css6    equ     css3            ;Same Checksum Vector Size
xlt6    equ     xlt3            ;Same Translate Table
;               ENDEF
;
;       Uninitialized Scratch Memory Follows:
;
begdat  equ     offset $        ;Start of Scratch Area
dirbuf  rs      128             ;Directory Buffer
alv0    rs      als0            ;Alloc Vector
csv0    rs      css0            ;Check Vector
alv1    rs      als1            ;Alloc Vector
csv1    rs      css1            ;Check Vector
alv2    rs      als2            ;Alloc Vector
csv2    rs      css2            ;Check Vector
alv3    rs      als3            ;Alloc Vector
csv3    rs      css3            ;Check Vector
alv4    rs      als4            ;Alloc Vector
csv4    rs      css4            ;Check Vector
alv5    rs      als5            ;Alloc Vector
csv5    rs      css5            ;Check Vector
alv6    rs      als6            ;Alloc Vector
csv6    rs      css6            ;Check Vector
enddat  equ     offset $        ;End of Scratch Area
datsiz  equ     offset $-begdat ;Size of Scratch Area
        db      0               ;Marks End of Module
```

APPENDIX I
PROGRAM LISTING OF INTELDSK.A86

```
;Prog Name  : INTELDSK.A86 (MDS S. Density Floppy Routines)
;Date       : 9 Aug 1982
;Written by : Jim John, SMC 1277, 649-0592
;Modified by: Tom V. Almquist and David Stevens
;For        : Thesis (AEGIS Modeling Group)
;Advisor    : Professor M.L. Cotton
;Purpose    : This code is an include file w/in CPMBIOS.A86.
;             It contains the routines for using the MDS
;             Single Density Floppy Disk.  It is configured
;             for a single iSCB 86/12A and does not use
;             common memory for I/O.

;------------------------EQUATES-------------------------

;port addresses

base            equu 078h         ;iSBC201 port address base
rrtport         equ base+1        ;read result type (input)
rrbport         equ base+3        ;read result byte (input)
resport         equ base+7        ;reset iSBC201 (output)
dstport         equ base          ;read subsystem status
                                  ;(input)
ialport         equ base+1        ;write iopb addr low
                                  ;(output)
iahport         equ base+2        ;write iopb addr high
                                  ;(output)

;command codes & masks

rdcode          equ 4             ;read command code
wrcode          equ 6             ;write command code
cwcode          equ 80H           ;channel command code
intbit          equ 04h           ;interrupt bit mask

retries         equ 10            ;for disk i/o, before error


;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                 ENTRY POINT ROUTINES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;---------------------------------------------------------
inteldsk_init:  ;initialize disk controller
                ;actually done by iSBC86/12 monitor

        ret
```

```
;-----------------------------------------------------------
inteldsk_home:
        ret

;-----------------------------------------------------------
inteldsk_seldsk:
        ret

;-----------------------------------------------------------
inteldsk_settrk:
        ret

;-----------------------------------------------------------
inteldsk_setsec:
        ret


;-----------------------------------------------------------
inteldsk_read:  ;read sector from disk

        mov cl,4
        mov al,unit             ;combine disk selection
        sal al,cl               ;with opcode
        or al,rdcode            ;to make io command for read
        mov io_com,al           ;set it in comd word of iopb
        call dsk_io             ;and execute it
        ret

;-----------------------------------------------------------
inteldsk_write: ;write to disk

        mov cl,4                ;create io command for write
        mov al,unit
        sal al,cl
        or al,wrcode
        mov io_com,al
        call dsk_io             ;go do it
        ret


;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                    SUBROUTINES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;-----------------------------------------------------------
dsk_io:         ;execute disk read or write function for
                ;iSBC201 controller.  Sets up remainder of
                ;iopb and sends its addr to the controller
                ;then polls for a response and checks for
                ;error conditions.

        mov io_chw,cwcode      ;set no wait code for channel
```

```
        mov io_nsc,1                ;transfer 1 sector
        mov al,sector               ;set up iopb trk and sect
        mov io_sec,al
        mov al,track
        mov io_trk,al
        mov cl,4                    ;recombine dma seg and addr
        mov ax,dma_seg
        sal ax,cl
        add ax,dma_adr
        mov io_adh,ah               ;set it in addr word of iopb
        mov io_adl,al
        mov try_cnt,retries
dio1:   in al,rrtport               ;clear controller
        in al,rrbport
        mov cl,4                    ;get address of iopb
        mov ax,cs
        sal ax,cl
        add ax,offset iopb
        out ialport,al              ;and send it out
        mov cl,8
        sar ax,cl
        out iahport,al
dio2:   in al,dstport               ;wait for contrler interrupt
        and al,intbit
        jz dio2
        in al,rrtport               ;check completion code
        or al,al
        jz dio3
        in al,rrbport               ;status chgd, ignore result
        jmps dio4                   ;and retry
dio3:   in al,rrbport               ;check io result
        or al,al
        jz dio6                     ;ret with al=0 if no error
dio4:                               ;error if we got here
        dec try_cnt                 ;decmt count and try again
        jnz dio1                    ;try again if any left
        or al,error                 ;set permanent error code
dio6:   ret

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                  PRIVATE DATA AREA
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

iopb    rb 7                ;i/o parameter block
io_chw  equ iopb            ;iopb channel byte
io_com  equ iopb + 1        ;command byte
io_nsc  equ iopb + 2        ;sectors to xfer (always 1)
io_trk  equ iopb + 3        ;selected track
io_sec  equ iopb + 4        ;selected sector
io_adl  equ iopb + 5        ;physical address for SBC201 DMA
io_adh  equ iopb + 6
try_cnt rb 1                ;disk error retry counter
```

APPENDIX J
PROGRAM LISTING OF LDCPM.A86

```
;Prog Name     : LDCPM.A86
;Written by    : T.V. Almquist and D. Stevens
;              : This program reads the file entitled
;              : CPMSLAVE.CMD into common memory beginning at
;              : location E000:500.


    cseg
        org     0100h
        jmp     start

;**********************************************************
;                       Equates
;**********************************************************

   cr           equ     0dh     ;carriage return
   lf           equ     0ah     ;line feed
   drive        equ     0004h   ;target CP/M drive #
   bdos_int     equ     224     ;interupt vector
   pstrf        equ     9       ;print string function
   seldskf      equ     14      ;select disk function
   openf        equ     15      ;open file function
   readf        equ     20      ;read function
   dmaf         equ     26      ;set dma offset function
   dmabf        equ     51      ;set dma base function


;**********************************************************
;                    Subroutines
;**********************************************************

    seldisk:                    ;select target disk #

        mov     cl,seldskf
        mov     dx,drive
        jmp     sys_vec

;----------------------------------------------------------
    openfnc:                    ;open file denoted in fcb

        mov     cl,openf
        mov     dx,offset fcb
        jmp     sys_vec

;----------------------------------------------------------
```

```
setdmab:                            ;set dma base address

        mov     cl,dmabf
        jmp     sys_vec

;-------------------------------------------------------------
setdma:                             ;set dma offset

        mov     cl,dmaf
        jmp     sys_vec

;-------------------------------------------------------------
read:                               ;read 128 bytes from file
                                    ;in fcb

        mov     dx,offset fcb
        mov     cl,readf
        jmp     sys_vec

;-------------------------------------------------------------
msg:                                ;print a character string
                                    ;end of string denoted by 0

        mov     cl,pstrf
        jmp     sys_vec

;-------------------------------------------------------------
sys_vec:                            ;execute bdos function call

        int     bdos_int
        ret

;**************************************************************
;                       Main Program
;**************************************************************

start:
        call    seldisk             ;select desired disk
        call    openfnc             ;open file
        cmp     al,255              ;if file not found
        jne     cont
        mov     dx,offset nofile
        call    msg                 ;print error msg
        jmp     stop

cont:
        mov     dx,cs               ;save 1st page in local
        call    setdmab             ;memory
        mov     dx,offset page1
        call    setdma
        call    read                ;read 1st page
```

```
        ;read file into commom memory

        mov     dx,0e000h           ;set dma base to common
        call    setdmab             ;memory
        mov     dx,0500h            ;desired offset

    readfile:
        call    setdma
        push    dx
        call    read                ;read 128 byte page
        cmp     al,01h              ;read complete ?
        je      done
        cmp     al,00h              ;repeat
        je      contread
        mov     dx,offset rerr      ;otherwise print read error
        call    msg
        jmp     stop

    contread:
        pop     dx
        add     dx,080h             ;increment dma offset for
        jmp     readfile            ;next page

    done:
        mov     dx,offset fmsg      ;print completion msg
        call    msg

stop:
        mov     cl,00h              ;return to CP/M
        mov     dl,00h
        int     bdos_int

;************************************************************
;                          Data
;************************************************************

nofile  db      cr,lf,'CPMSLAVE.CMD Not Found On This Disk$'
rerr    db      cr,lf,'Read Error$'
fmsg    db      cr,lf,'CPMSLAVE.CMD Loaded into Common '
        db      'Memory$'

fcb     db      04,'CPMSLAVE','CMD',0,0,0,0,0,0,0,0,0,0,0,0
        db      0,0,0,0,0,0,0,0,0
page1   rs      128
        db      0
        end
```

APPENDIX K
PROGRAM LISTING OF LDBOOT.A86:

```
;Prog Name      : LDBOOT.A86
;Written by     : T.V. Almquist and D. Stevens
;               : This program loads the boot loader into
;               : common memory and is used by slave 86/12As.

;**********************************************************

    cseg
        org     0100h
        jmp     start

;**********************************************************
;                       Equates
;**********************************************************

   cr           equ     0dh     ;carriage return
   lf           equ     0ah     ;line feed
   drive        equ     0004h   ;target CP/M drive #
   bdos_int     equ     224     ;interupt vector
   pstrf        equ     9       ;print string function
   seldskf      equ     14      ;select disk function
   openf        equ     15      ;open file function
   readf        equ     20      ;read function
   dmaf         equ     26      ;set dma offset function
   dmabf        equ     51      ;set dma base function

;**********************************************************
;                     Subroutines
;**********************************************************

    seldisk:                    ;select target disk #

        mov     cl,seldskf
        mov     dx,drive
        jmp     sys_vec

;----------------------------------------------------------
    openfnc:                    ;open file denoted in fcb

        mov     cl,openf
        mov     dx,offset fcb
        jmp     sys_vec

;----------------------------------------------------------
```

```
    setdmab:                        ;set dma base address

        mov     cl,dmabf
        jmp     sys_vec

;------------------------------------------------------------
    setdma:                         ;set dma offset

        mov     cl,dmaf
        jmp     sys_vec

;------------------------------------------------------------
    read:                           ;read 128 bytes from file
                                    ;in fcb

        mov     dx,offset fcb
        mov     cl,readf
        jmp     sys_vec

;------------------------------------------------------------
    msg:                            ;print a character string
                                    ;end of string denoted by 0

        mov     cl,pstrf
        jmp     sys_vec

;------------------------------------------------------------
    sys_vec:                        ;execute bdos function call

        int     bdos_int
        ret

;************************************************************
;                   Main Program
;************************************************************

start:
        call    seldisk             ;select desired disk
        call    openfnc             ;open file
        cmp     al,255              ;if file not found
        jne     cont
        mov     dx,offset nofile
        call    msg                 ;print error msg
        jmp     stop

    cont:
        mov     dx,cs               ;save 1st page in local
        call    setdmab             ;memory
        mov     dx,offset page1
        call    setdma
        call    read                ;read 1st page
```

```
        ;read file into commom memory

        mov     dx,0e000h          ;set dma base to common
        call    setdmab            ;memory
        mov     dx,0400h           ;desired offset

    readfile:
        call    setdma
        push    dx
        call    read               ;read 128 byte page
        cmp     al,01h             ;read complete ?
        je      done
        cmp     al,00h             ;repeat
        je      contread
        mov     dx,offset rerr     ;otherwise print read error
        call    msg
        jmp     stop

    contread:
        pop     dx
        add     dx,080h            ;increment dma offset for
        jmp     readfile           ;next page

    done:
        mov     dx,offset fmsg     ;print completion msg
        call    msg

stop:
        mov     cl,00h             ;return to CP/M
        mov     dl,00h
        int     bdos_int

;**************************************************************
;                         Data
;**************************************************************

nofile  db      cr,lf,'BOOT.CMD Not Found On This Disk$'
rerr    db      cr,lf,'Read Error$'
fmsg    db      cr,lf,'BOOT.CMD Loaded into Common Memory$'
fco     db      04,'BOOT    ','CMD',0,0,0,0,0,0,0,0,0,0,0
        db      0,0,0,0,0,0,0,0,0
page1   rs      128
        db      0
        end
```

APPENDIX L
PROGRAM LISTING OF BOOT.A86

```
;Prog Name      : BOOT.A86
;Written by     : T.Almquist and D. Stevens
;Date           : 16 October 1982
;               : This program is the boot loader used by
;               : slave 86/12As to load CP/M-86.

;*************************************************************
;                        Equates
;*************************************************************

load_addr equ 0400h
cpm_addr  equ 0500h

;*************************************************************
;                     Main Program
;*************************************************************

cseg

        call    request             ;get ticket number
        mov     ax,0040h            ;set es to CP/M segment #
        mov     es,ax
        mov     di,0000h            ;set desired offset
        mov     ax,0e000h           ;set ds to common memory
        mov     ds,ax               ;segment #
        mov     si,cpm_addr         ;CPM.SLAVE offset
        mov     cx,1a00h            ;number of bytes to move
        cld                         ;from common memory to
    rep movs    ax,ax               ;local memory
        call    release             ;increment server #
        jmpf    dword ptr bios_offset + load addr
                                    ;transfer to CP/M

;*************************************************************
;                      Include File
;*************************************************************

        include sync.a86            ;for sharing common memory

;*************************************************************
;                          Data
;*************************************************************

bios_offset     dw      2500h   ;CP/M jump vector
bios_seg        dw      0040h
```

```
db      0
end
```

APPENDIX M
PROGRAM LISTING OF LOGIN.A86

```
;Prog Name   : LOGIN.A86
;Date        : 15 October 1982
;Written by  : T. Almquist and D. Stevens
;            : This program contains the code necessary to
;            : permit only one user at a time to be logged
;            : on to any I/O storage device.

;***********************************************************
;                        Equates
;***********************************************************

        busy   equ  0ffh           ;busy indicator
        ndsks  equ  7              ;number of CP/M disks

;***********************************************************
;                      Subroutines
;***********************************************************

        cseg $

login:

        push es                    ;set up to address common
        mov  ax,cmemseg            ;memory
        mov  es,ax
    log0:
        mov  bx,offset logmsg2     ;get console number
        call pmsg
        call conin                 ;ret console number in al
        cmp  al,31h                ;ensure response is between
        jl   log0                  ;1 and 4
        cmp  al,34h
        jg   log0
        mov  console,al            ;save console number
    log1:
        mov  bx,offset logmsg1     ;inital login msg
        call pmsg                  ;print message
        call conin                 ;get login disk
        cmp  al,41h                ;within range defined by
        jl   log1                  ;CPMMAST.DEF
        cmp  al,40h + ndsks        ;greater than g:
        jg   log1
        and  al,0fh                ;strip upper nibble
        sub  al,1                  ;normalize to zero
        mov  user,al               ;save login user disk
```

```
        ;determine if disk is free

        xor  bx,bx
        mov  bl,al                  ;set up to index logtbl
        mov  al,busy
        lock xchg al,logtbl[bx]
        test al,al                  ;is disk free?
        jz   log2                   ;if so, enter console #
        cmp  al,console             ;is console already logged
        jnz  restore                ;if not, restore logtbl
    log2:
        xor  bx,bx                  ;clear bx
        mov  bl,user                ;offset in logtbl
        mov  al,console
        lock xchg al,logtbl[bx]     ;enter console number
        jmp  log_ret

   restore:
        lock xchg al,logtbl[bx]     ;restore logtbl entry
        mov  bx,offset logmsg3      ;request another disk #
        call pmsg
        jmp  log1
   log_ret:
        pop  es
        ret

;-----------------------------------------------------------

init_login:                         ;initialize logtbl entries

        push es                     ;address common memory
        mov  ax,cmemseg
        mov  es,ax
        xor  bx,bx
        xor  cx,cx
        mov  cl,ndsks               ;entry for each disk
   again:
        mov  logtbl[bx],0           ;initialize elements of
        inc  bx                     ;logtbl to 0
        loop again
        pop  es
        ret

;***********************************************************
;                          Data Area
;***********************************************************

user            rb  1
console         rb  1
logmsg1         db  cr,lf,'Enter Login Disk Letter (A,D,E,F,G)'
                db  cr,lf,0
logmsg2         db  cr,lf,'Enter Console Number (1,2,3,4)'
```

```
                db  cr,lf,0
logmsg3         db  cr,lf,'Disk in Use ---- Reselect',cr,lf,0

                    eseg
                    org 20h

logtbl              rb  ndsks          ;allot memory for logtbl

                    cseg $

;end login.a86
```

APPENDIX N
PROGRAM LISTING OF SYNC.A86

```
;Prog Name      :Synch.A86
;Date           :7 October 1982
;Written by     :Nick Hammond
;Modified by    :T. Almquist and D. Stevens
;For            :Thesis
;Advisor        :Professor Kodres
;Purpose        :Provide synchronizations of CPM/86 read
;                and write operations to the MBB-80 bubble
;                memory board and the REMEX Data Warehouse.
;
;
;
;*************************************************************
;                    Synchronization Routine
;*************************************************************


;*************************************************************
;                          Equates
;*************************************************************

       cmemseg equu 0e000h       ;segment address of
                                 ;common memory
       dcount  equ 100           ;bus contention time delay

;*************************************************************
;                        Subroutines
;*************************************************************

       cseg $

ticket:                     ;return the next ticket number in
                            ;bx

       xor  ax,ax                ;set reserved value
       lock xchg ax,next         ;get ticket number
       test ax,ax
       jz   ticket               ;repeat if reserved
       mov  bx,ax                ;return next ticket
       inc  ax
       jnz  tic1
       inc  ax                   ;skip reserved value
tic1:  mov  next,ax              ;increment ticket number
```

```
        ret

;-------------------------------------------------------------
 await:                        ;wait for server number to match
                               ;the customers ticket number passed
                               ;in bx. To reduce bus contention, a
                               ;delay is used between periodic
                               ;checks of the server number

        cmp  bx,server                 ;if ticket = server
        je   awa2                      ;continue process
        mov  cx,dcount                 ;if not, insert delay
awa1:   dec  cx
        jnz  awa1
        jmp  await                     ;check server again
awa2:   ret

;-------------------------------------------------------------
 advance:                      ;increment server number to next
                               ;value

        inc  server                    ;server=server+1
        jnz  adv1
        inc  server                    ;skip reserved value
adv1:   ret

;-------------------------------------------------------------
 request:                      ;get a ticket number and wait to be
                               ;served

        push es
        mov  ax,cmemseg                ;set es to address common
        mov  es,ax                     ;memory
        call ticket                    ;get ticket number
        call await                     ;wait to be served
        pop  es
        ret

;-------------------------------------------------------------
 release:                      ;adv server number on completion
                               ;of read or write operation

        push es
        mov  ax,cmemseg                ;set es to address common
        mov  es,ax                     ;memory
        call advance                   ;inc server number
        pop  es
        ret

;-------------------------------------------------------------

 initsync:                     ;initialize sequencer variables
```

```
        push es
        mov  ax,cmemseg            ;set es to address common
        mov  es,ax                 ;memory
        mov  ax,1                  ;server=next=1
        mov  server,ax
        mov  next,ax
        pop  es
        ret

;***********************************************************
;                          Data
;***********************************************************

        eseg               ;only one set of sequencer variables
                           ;exist in common memory; accessed
                           ;via es

        server  rw 1
        next    rw 1

        cseg $

;end synch.a86
```

LIST OF REFERENCES

1. Candalor, M. B., Alteration of the CP/M-86 Operating System, Masters Thesis, Naval Postgraduate School, Monterey California, 1981.

2. Hicklin, M. S. and J. A. Neufeld, Adaptation of Magnetic Bubble Memory in a Standard Microcomputer Environment, Masters Thesis, Naval Postgraduate School, Monterey California, 1981.

3. Hammond, Nick, Sharing of a Peripheral Device Between Processors, CS 3500 Project, Naval Postgraduate School, Monterey, CA, 1982.

4. Digital Research, ASM-86: The CP/M-86 Assembler User's Guide, 1981.

5. Digital Research, DDT-86: The CP/M-86 Dynamic Debugging Tool for the 8086 User's Guide, 1981.

6. Digital Research. CP/M-86 Operating System Guide. 1981.

7. EX-CELL-O Corporation, REMEX Technical Manual for Data Warehouse Models RDW 3100, RDW 3200, 1979.

8. EX-CELL-O Corporation, REMEX Technical Manual for the Multibus Interface Assy 614415-001, 1980.

9. Intel Corporation, iCS 80 Industrial Chassis Hardware Reference Manual, 1979.

10. Intel Corporation, iSBC 604/614 Hardware Reference Manual, 1979.

11. Intel Corporation, iSBC 640 Power Supply Hardware Reference Manual, 1979.

12. Reed, D.P. and Kanodia, R.K., Synchronization with Event Counters and Sequencers, CACM VOL 22, NO 2, 1979.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Defense Logistic Studies Information Exchange U. S. Army Logistics Manaement Center Fort Lee, Virginia 23801	1
3.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5.	Associate Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
6.	Lcdr. Ronald Modes, USN, Code 52Mf Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
7.	Cdr. John Pfeiffer, USN, Code 37 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
8.	Lcdr. Thomas V. Almquist, USN 12555 McIntire Court Woodbridge, Virginia 22192	2
9.	CPT David S. Stevens, USA 2205 Deckman Lane Silversprings, Maryland 20906	2
10.	Daniel Green (Code N20E) Naval Surface Warfare Center Dahlgreen, Virginia 22449	1

11. CDR J. Donegan, USN 1
PMS 400B5
Naval Sea Systems Command
Washington, DC 20362

12. RCA AEGIS Data Repository 1
RCA Corporation
Government Systems Division
Mail Stop 127-327
Moorestown, New Jersey 08057

13. Library (Code E33-05) 1
Naval Surface Warfare Center
Dahlgreen, Virginia 22449

14. G. Luke 1
Fleet Systems Department
Applied Physics Laboratory
Laurel, Maryland 20810

15. Robert Coates 1
5840 Avienda Jinette
Bonsall, California 92003

END

FILMED

5-83

DTIC